Embedded Linux Conference

Effectively Measure and Reduce Kernel Latencies for Real-time Constraints

Embedded Linux Conference 2017

Jim Huang <jserv.tw@gmail.com>, Chung-Fan Yang <sonic.tw.tp@gmail.com>

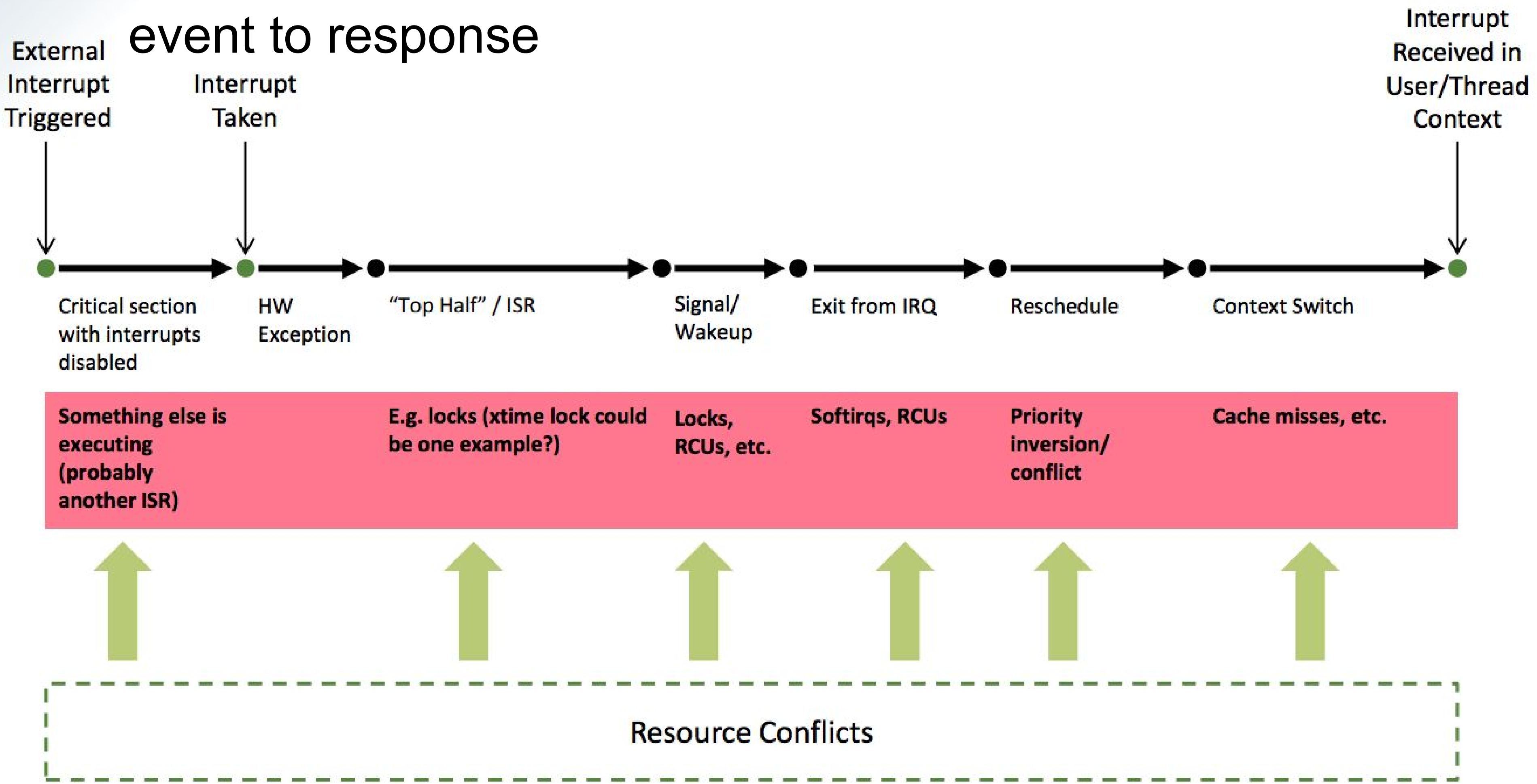
National Cheng Kung University, Taiwan

Goals of This Presentation

- The latency means the time after a task is invoked and before it is executed, depending on Linux scheduler latency, the deferred execution methods, and the priorities of competing tasks.
- Introduce new measurement tools by efficient ways to visualize system latency. (available on GitHub!)
- Major target: PREEMPT_RT (Locking primitives: spinlocks are replaced by RT Mutexes. Interrupt Handlers run in a kernel thread)
- Analyze and reduce the latency
    - ARM Cortex-A9 multi-core for case study

PREEMPT_RT in a nutshell

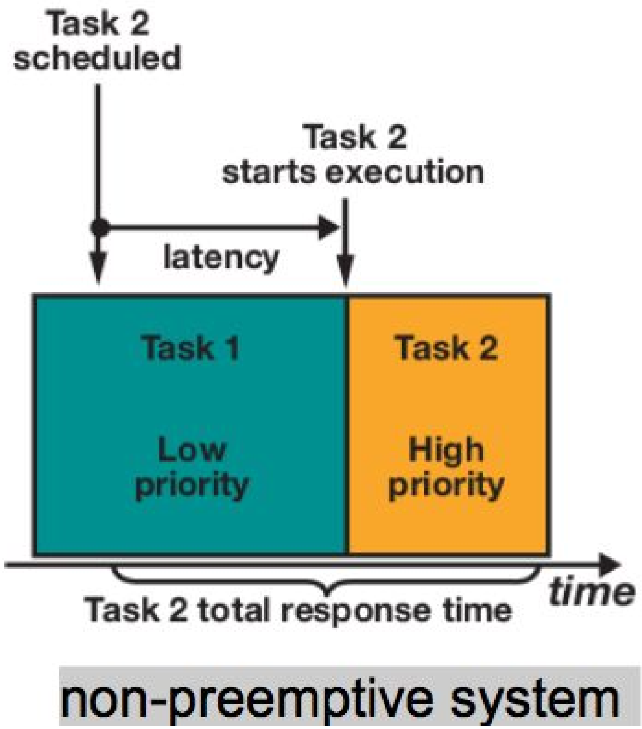
- Minimize Linux Interrupt Processing Delays from external event to response

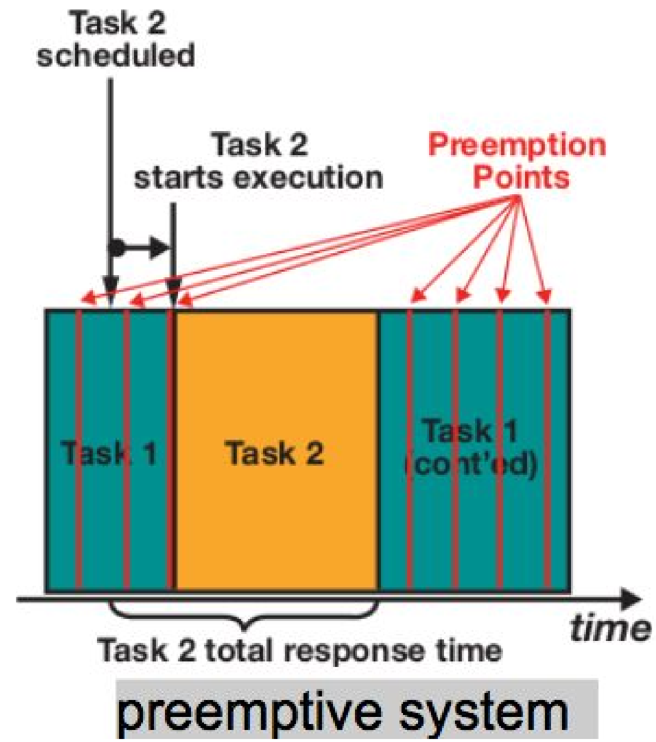


Source: 4 Ways to Improve Performance in Embedded Linux Systems, Michael Christofferson, Enea (2013)

Preemptive Kernel

- Controlling latency by allowing kernel to be preemptible everywhere
- Increase responsibility; decrease throughput

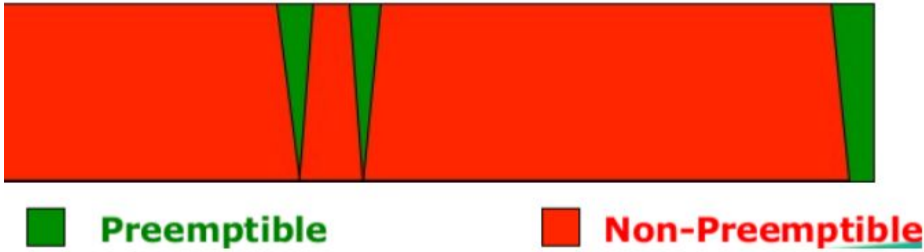




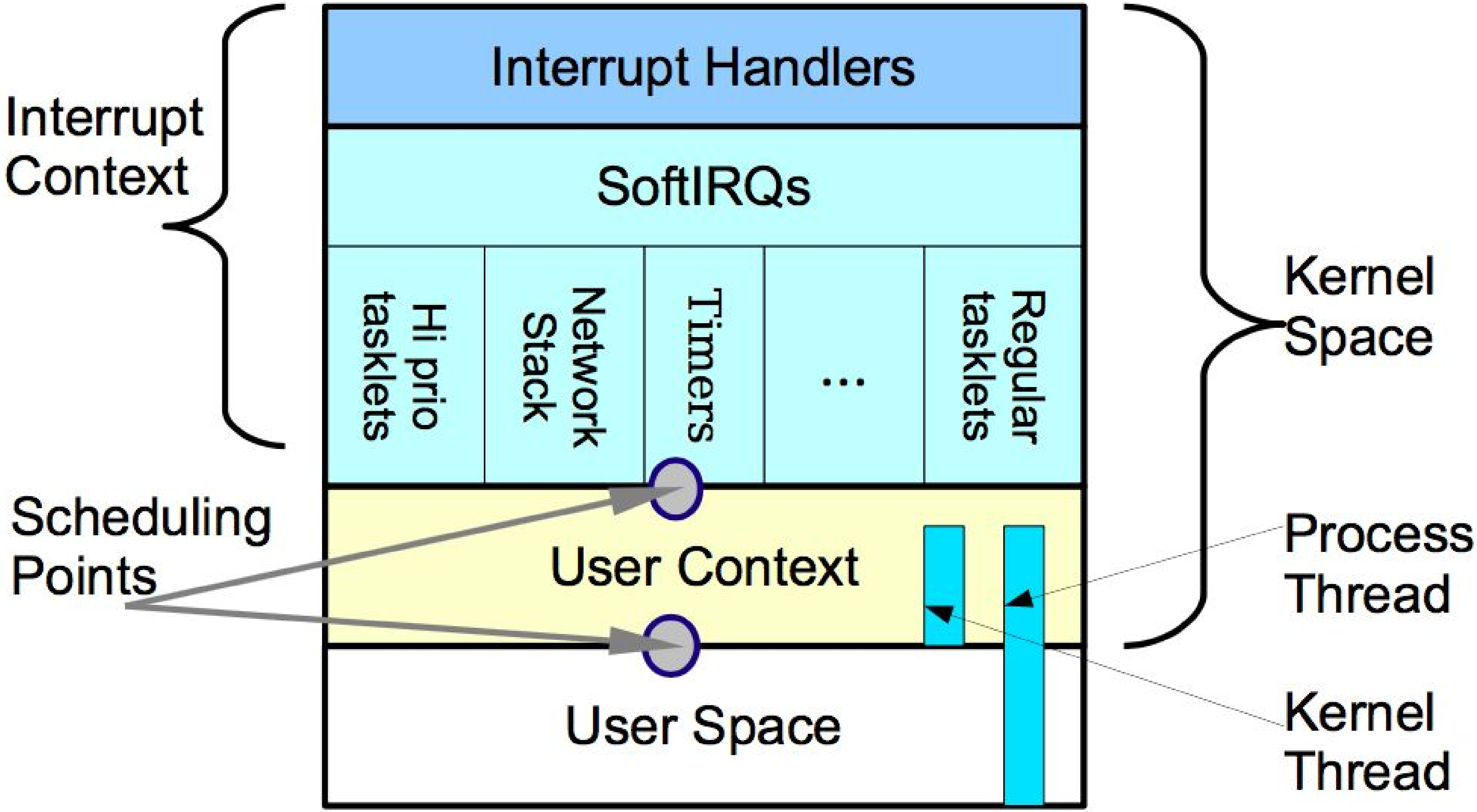
- preemption: the ability to interrupt tasks at many “preemption points”
- The longer the non-interruptible program units are, the longer is the waiting time of a higher priority task before it can be started or resumed.
- PREEMPT_RT makes system calls preemptible as well

Source: Understanding the Latest Open-Source Implementations of Real-Time Linux for Embedded Processors, Michael Roeder, Future Electronics

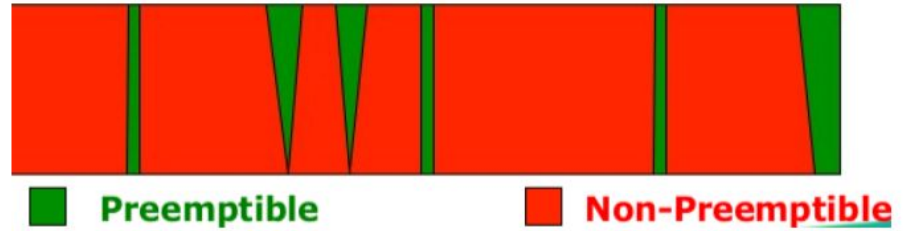
PREEMPT_NONE



Preemption is not allowed in Kernel Mode
Preemption could happen upon returning to user space

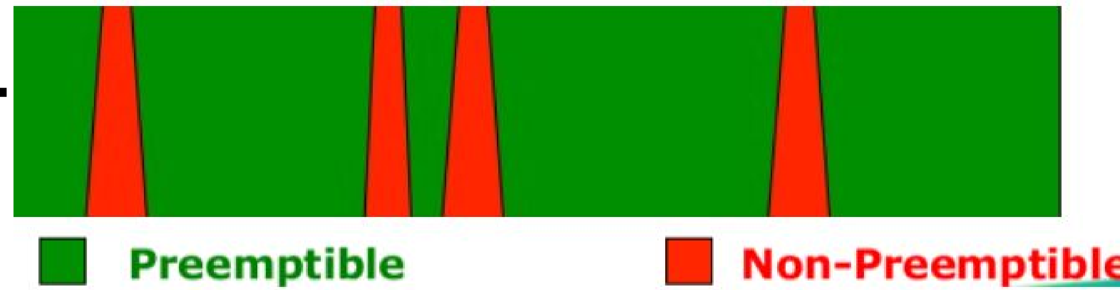


PREEMPT_VOLUNTARY

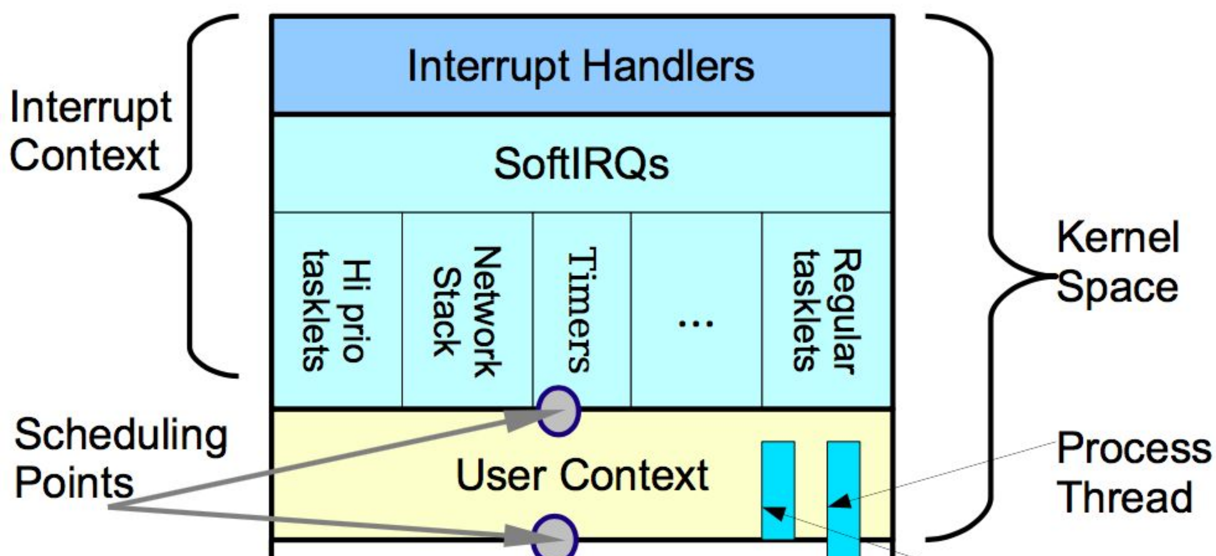


Insert explicit preemption point in Kernel: might_sleep
Kernel can be preempted only at preemption point

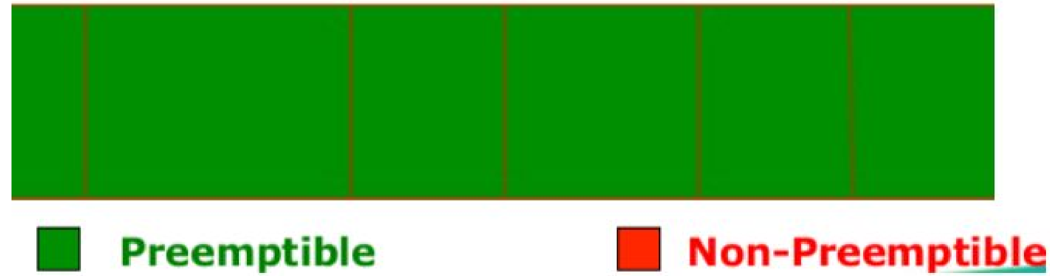
CONFIG_PREEMPT



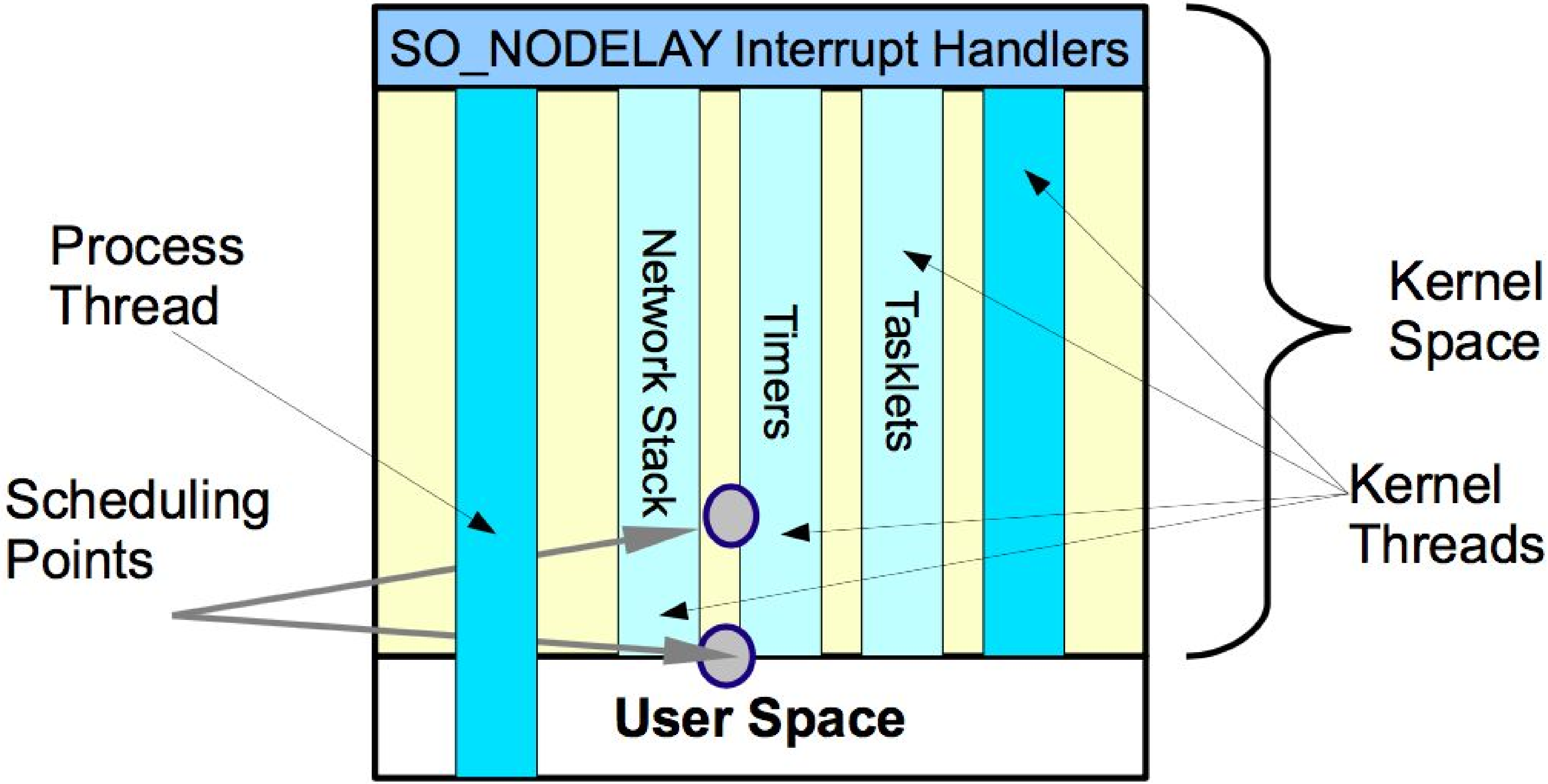
- Implicit preemption in Kernel
- preempt_count
  - Member of thread_info
  - Preemption could happen when preempt_count == 0



PREEMPT_RT_FULL:
Threaded Interrupts



Reduce non-preemptible cases in kernel: spin_lock, interrupt



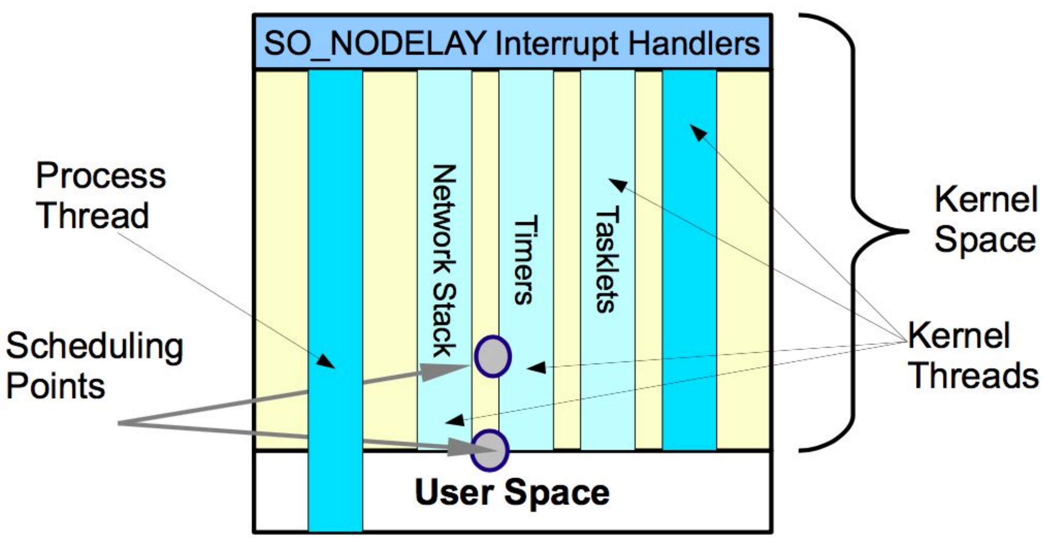
PREEMPT_RT Internals

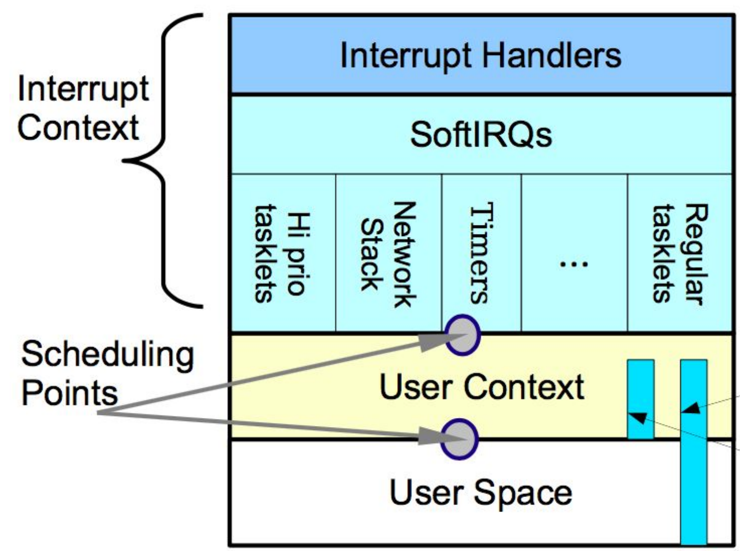
excellent talk "*Understanding a Real-Time System*" by Steven Rostedt

- softirq is removed
  - ksoftirqd as a normal kernel thread, handles all softirqs
  - softirqs run from the context of who raises them
- Exceptions: for softirqs raised by real hard interrupts
  - RCU invocation
  - timers

System Management Threads

- RCU
- Watchdog
- Migrate
- kworker
- ksoftirqd
- posixcputimer





PREEMPT_RT: Replace spin_lock_irqsave with spin_lock

include/linux/spin_lock.h

```
#ifdef CONFIG_PREEMPT_RT_FULL
# include
<linux/spinlock_rt.h>
#else /* PREEMPT_RT_FULL */
```

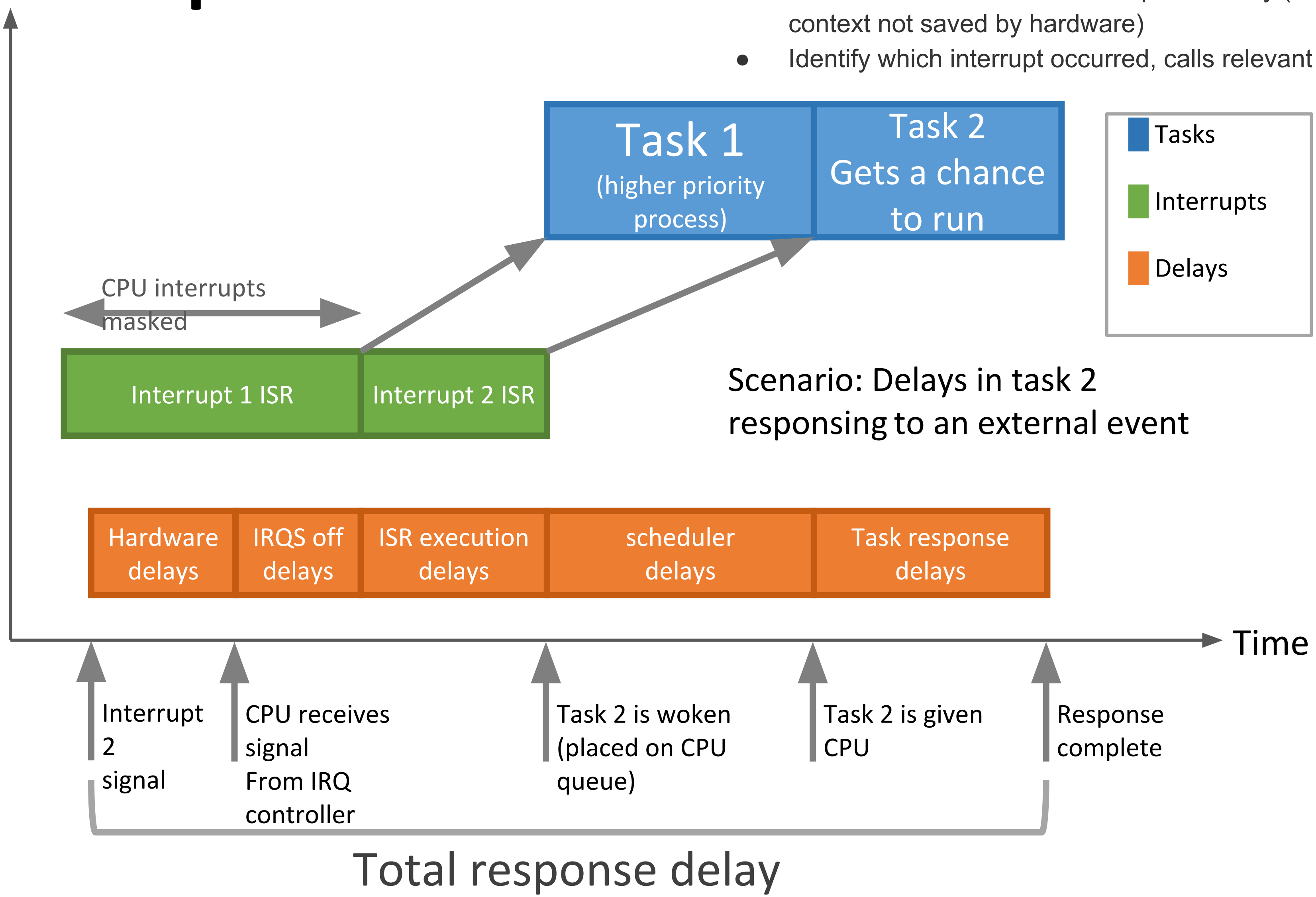
include/linux/spinlock_rt.h

```
#define spin_lock_irqsave(lock, flags) \
  do { \
    typecheck(unsigned long, flags); \
    flags = 0; \
    spin_lock(lock); \
  } while (0)
...
#define spin_lock(lock) \
  do { \
    migrate_disable(); \
    rt_spin_lock(lock); \
  } while (0)
```

interrupt handling in Linux

- Interrupt controller sends a hardware signal
- Processor switches mode, banking registers and disabling irq
- Generic Interrupt vector code is called
- Saves the context of the interrupted activity (any context not saved by hardware)
- Identify which interrupt occurred, calls relevant ISR

Latency Measurement: Wake up

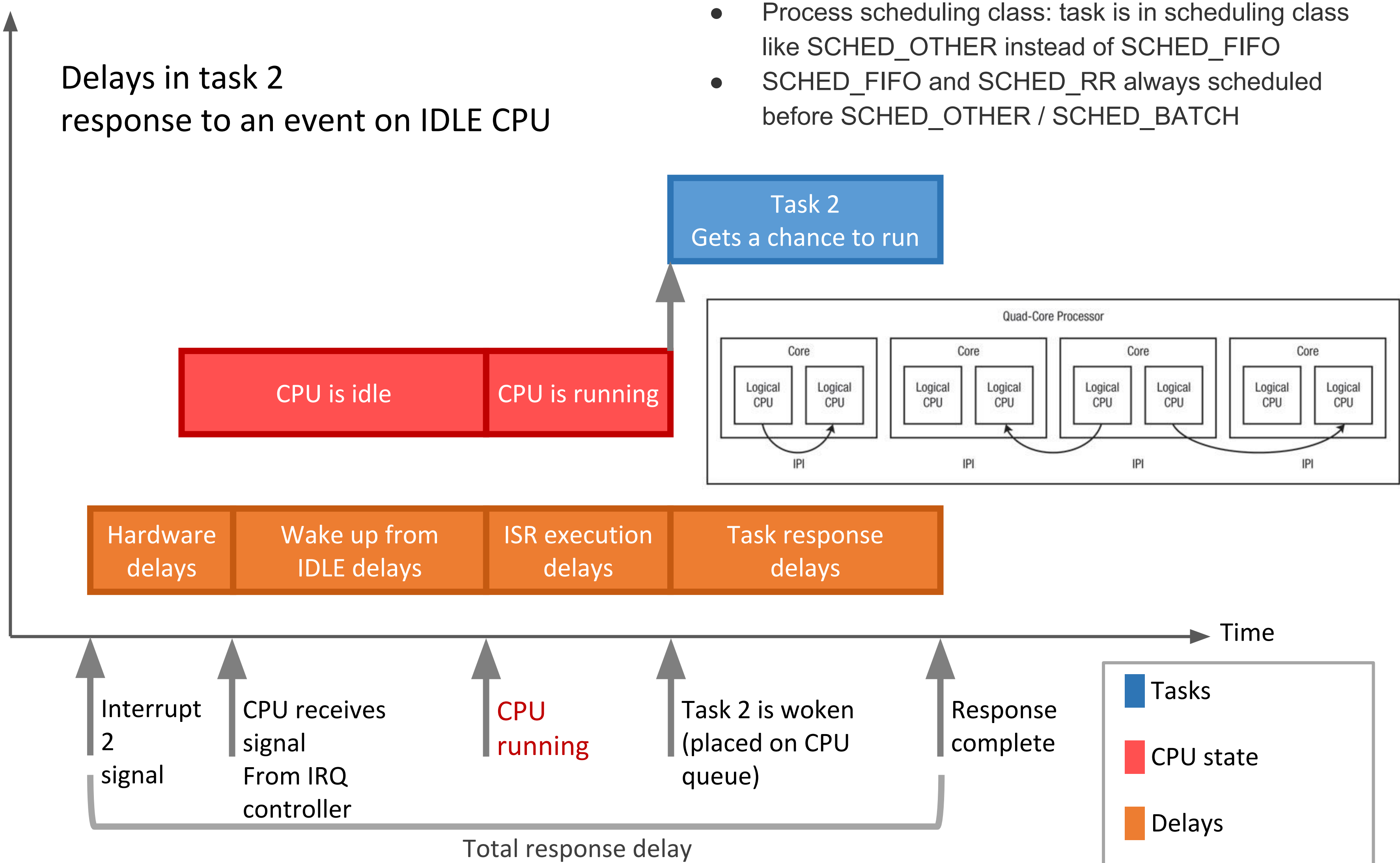


Latency Measurement: Wake up on IDLE CPU

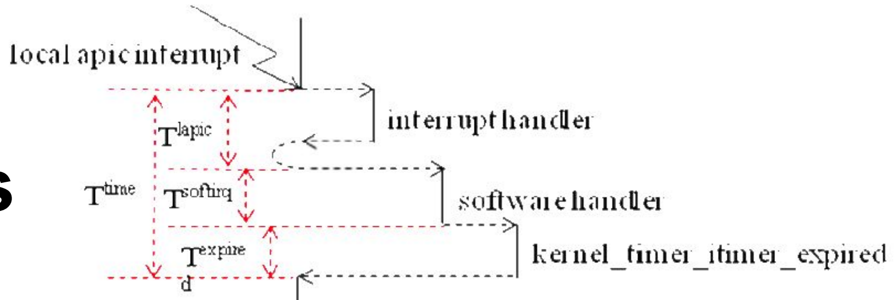
Scheduler needs to put woken up task on CPU, otherwise, latency increases.

Things preventing that:

- Process priority: Low prio task waits on the rq while high prio given cpu
- Process scheduling class: task is in scheduling class like SCHED_OTHER instead of SCHED_FIFO
- SCHED_FIFO and SCHED_RR always scheduled before SCHED_OTHER / SCHED_BATCH



Microscope Measurements



HRTIMER_SOFTIRQ is executed before softirq handler because higher prio task

- ## Clocksource and High Resolution Timer

  Accuracy of timer in Linux depends on the accuracy of hardware and software interrupts.
  Timer interrupts are not occurring accurately when the system is overloaded. It would cause timer latency in kernel

- ## Task switching cost

  Process switching cost is significantly larger than thread switching. Process switching needs to flush TLB.
  If RT application consists of lots of processes, process switching measurement is necessary

- ## Page faults

  Initial memory access causes page fault, and this causes more latency.
  Page-out to swap area also causes page faults. Use mlockall and custom memory allocators

- ## Multi-core

  tasks can move from local core to remote cores. This migration causes additional latency.
  Tasks can be fixed to a specific core by cpuset cgroup

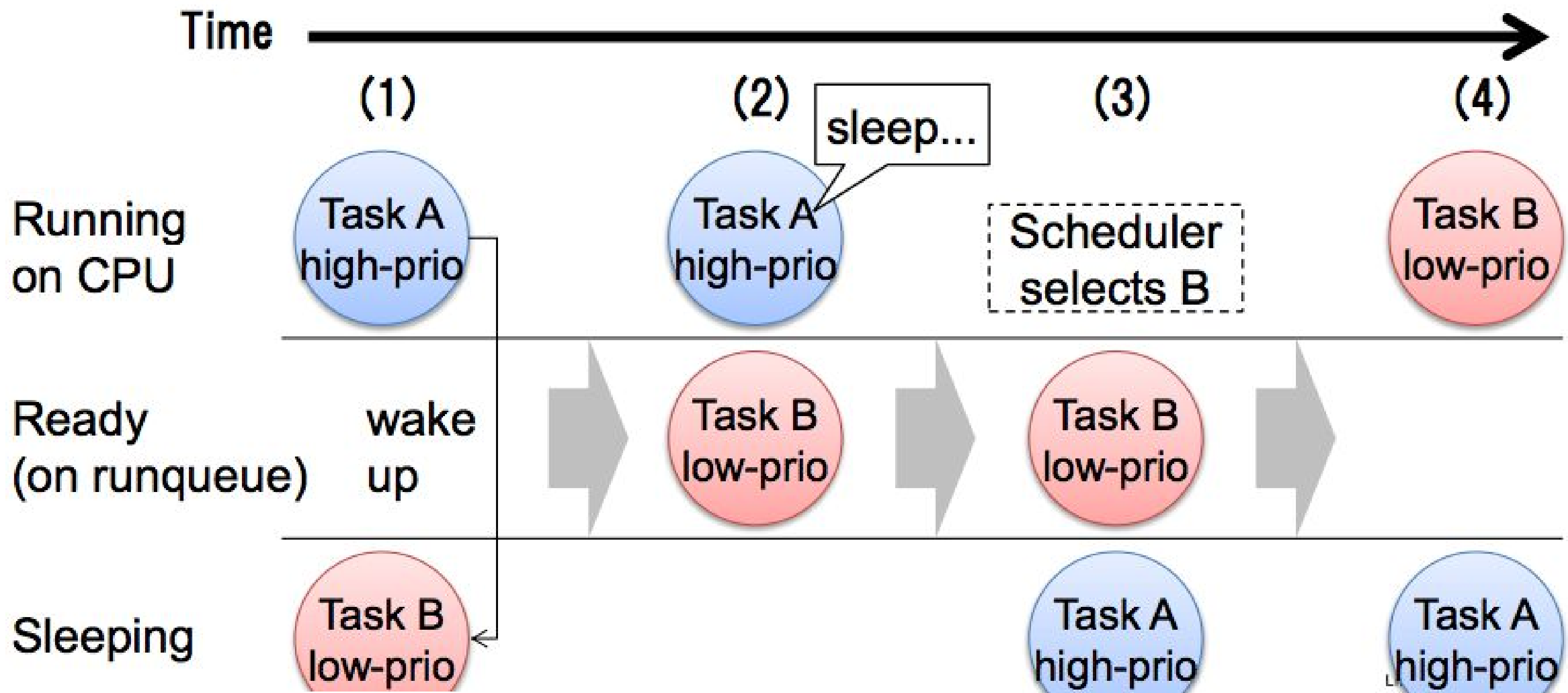
- ## Locks

  spin_locks are now mutexes, which can sleep. spin_locks must not be in atomic paths. That is, preempt_disable or local_irq_save. RT mutex uses priority inheritance, and no more futexes. cost gets higher in general.

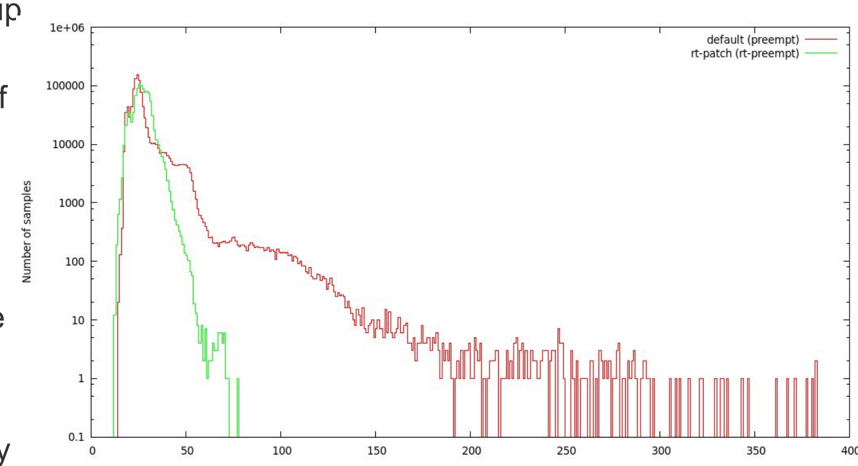
Before real measurements, prepare workload

- Hackbench
  - test scheduler and unix-socket (or pipe) performance by spawning processes and threads
- stress / stress-ng
  - stress tests and compare various
  - The normalized data is then summed to give an overall view of system impact each different kernel has on different types of metrics across a very wide range of stress tests.
- mctest
  - our in-house periodic task which evaluates robot control algorithms in real products.
  - Algorithms can be executed in both user and kernel mode.

General latency measurement



- cyclictest measures the delta from when it's scheduled to wake up from when it actually does wake up.
- Use HRT. The data gathered allows one to see the distribution of latencies from timer delays
- A long tail of latencies shows that some paths in the kernel are taking a while to be preempted during critical sections where the kernel cannot be interrupted.
- Disadvantage of histogram is the loss of timing information of the latency events, and there is no way to retrospectively gain information which task was preempted by which task and which phase of the preemption was responsible for the elevated latency



How cyclictest works

- measure latency of response to a stimulus
- sleep for a defined time
- measure actual time when woken up
- calculate difference of actual and expected time

```
while (!shutdown) {
     clock_nanosleep(&next);
    clock_gettime(&now);
    diff = calcdiff(now, next);
    next += interval;
}
```

Source: Real-Time Linux on Embedded Multi-Core Processors Andreas Ehmanns, MBDA Deutschland GmbH

More Tools for Measurements

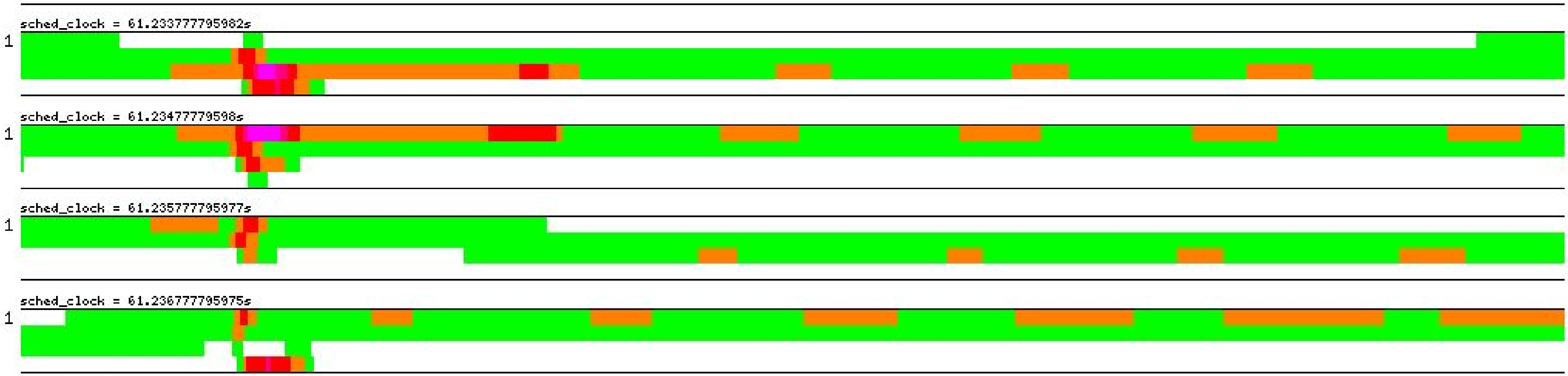
Profiling Tools

- Perf
  - Traditional way of understanding resource utilization
  - Samples CPU’s PMU periodiclly
  - Longer sampling period
  - Use statistical methods to estimate figures

Profiling Tools

- Sched Profiler
  - Proposed in paper “A Decade of Wasted Cores” (EuroSys 2016)
  - Patch the Linux scheduler and insert profiling points
  - Profiling points get executed every time
  - Capturing every scheduler stat change

Intel Core-i5 Gen-6th CPU running hackbench



- **Visualization: Heat Map**
  - Each line is a logic core
  - Each Pixel is 10us
  - Each line wrap is 10ms
- **By Default**
  - Profiles Number of items in Run Queue
  - Balance events
  - Task migration

>5 Tasks

4 Tasks

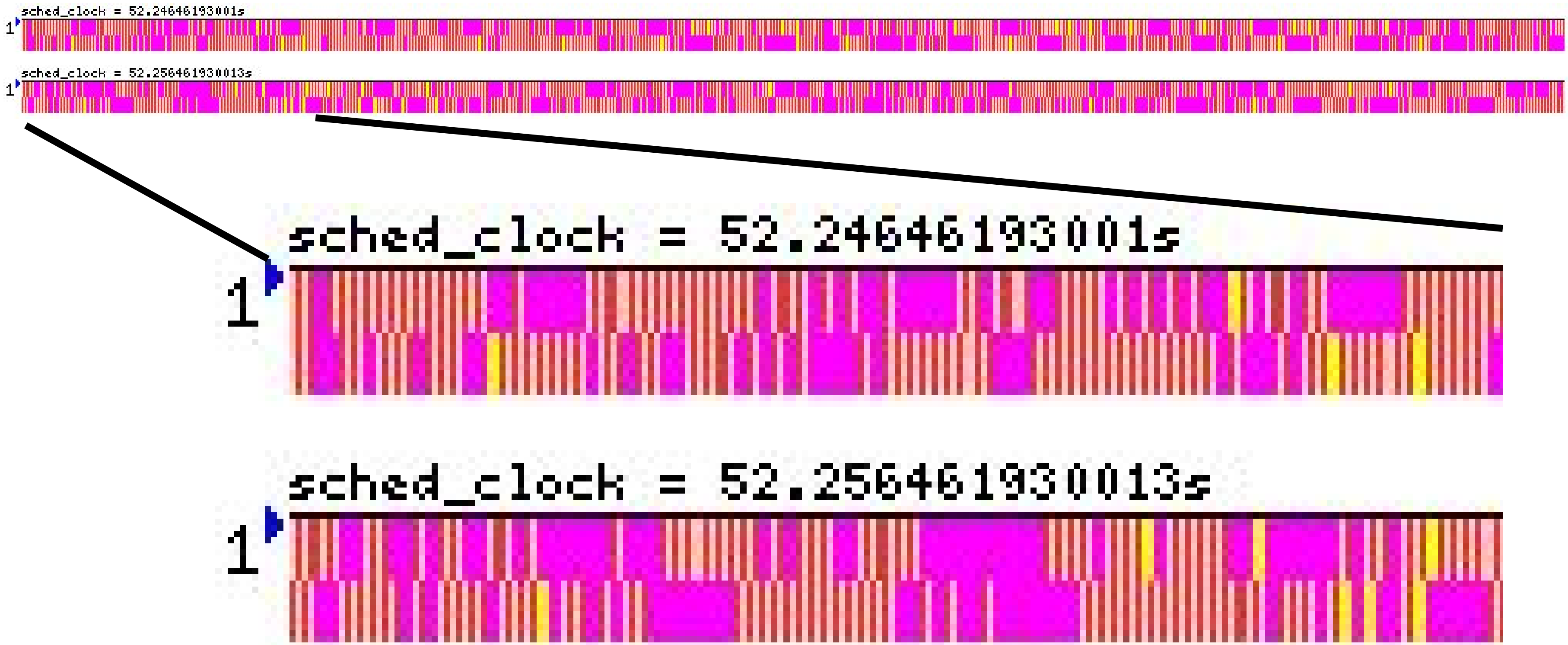
3 Tasks

2 Tasks

1 Task

CPU Idle

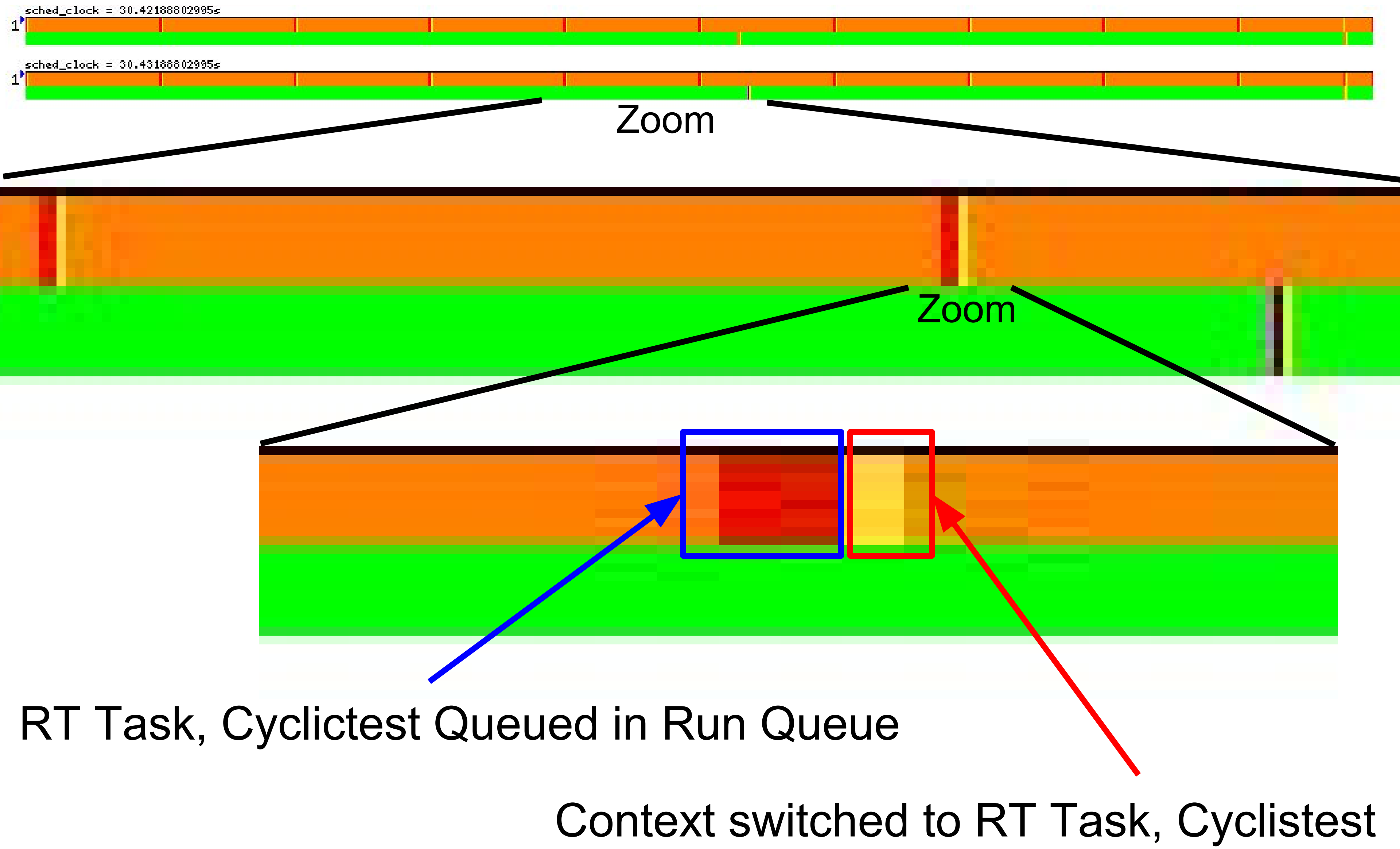
CTX points of Cortex-A9 running hackbench:



- **What we modified**
  - Keep the heat map
  - Profile the context switch time and switch-to PID
  - Plot the Point of context switches

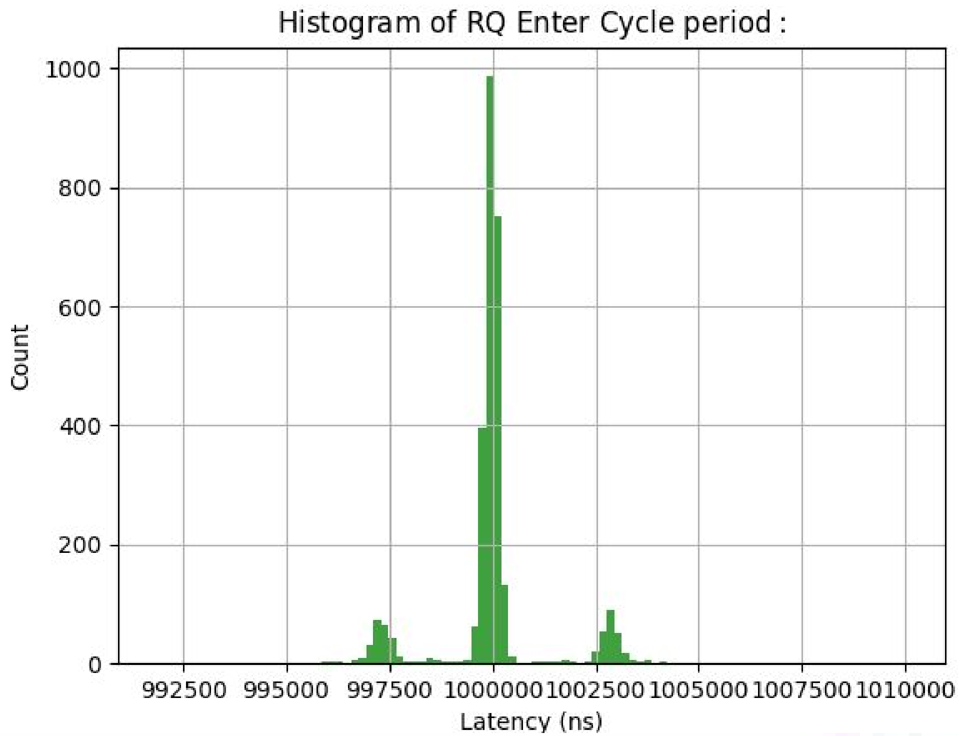
Context Switch

CTX points of Cortex-A9 running stress & cyclistest



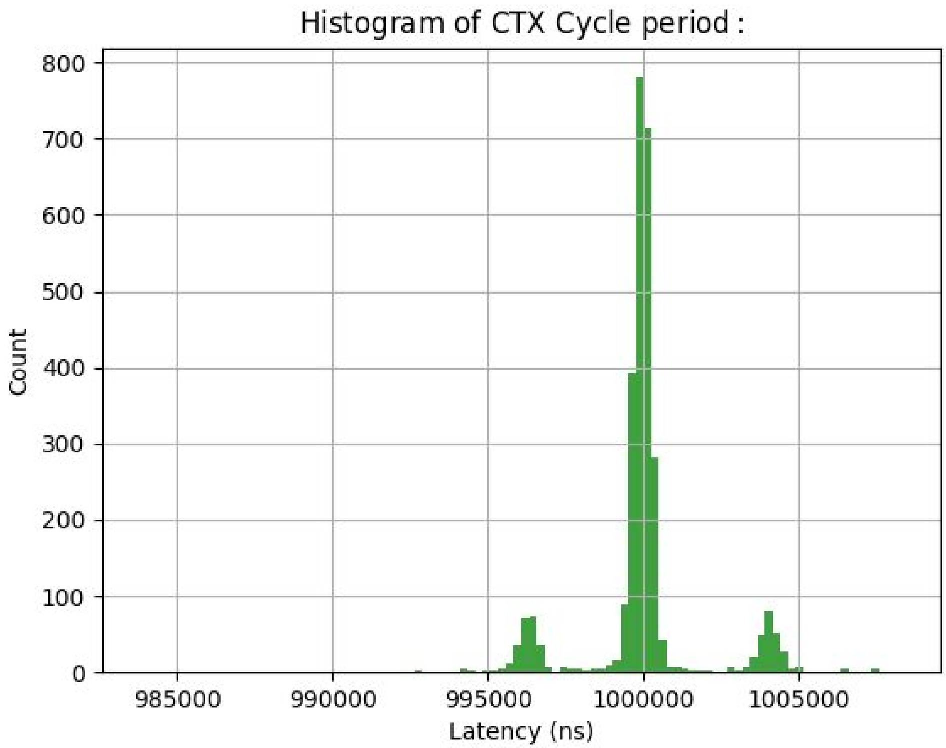
PREEMPR_RT Cortex-A9 running cyclictest at 1ms

The cycle time of RT task entering the Run Queue



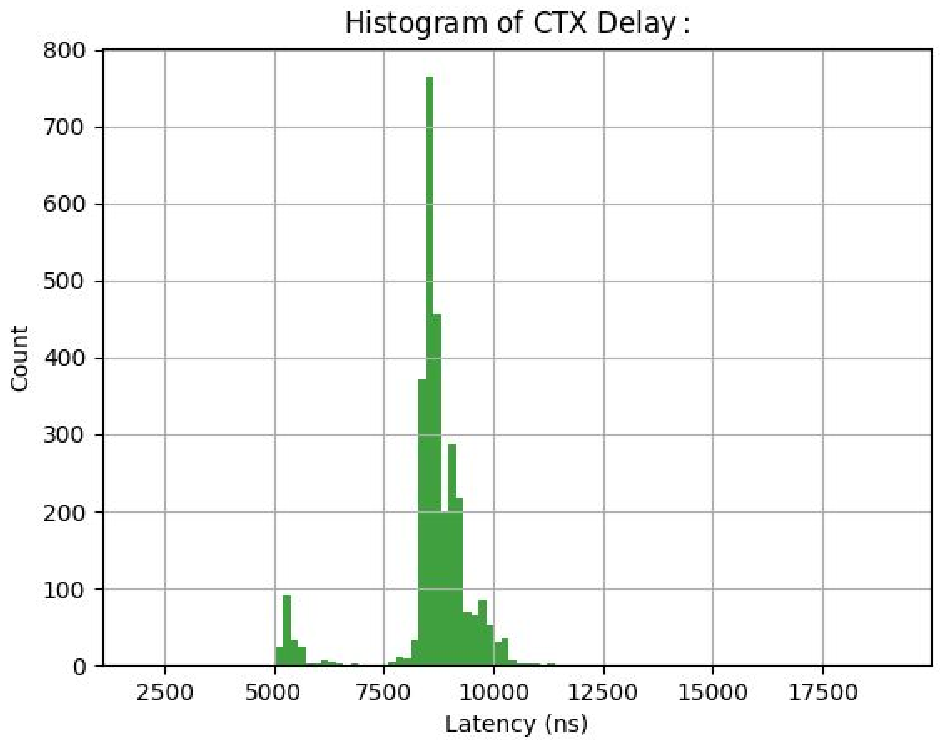
PREEMPT_RT Cortex-A9 running cyclictest

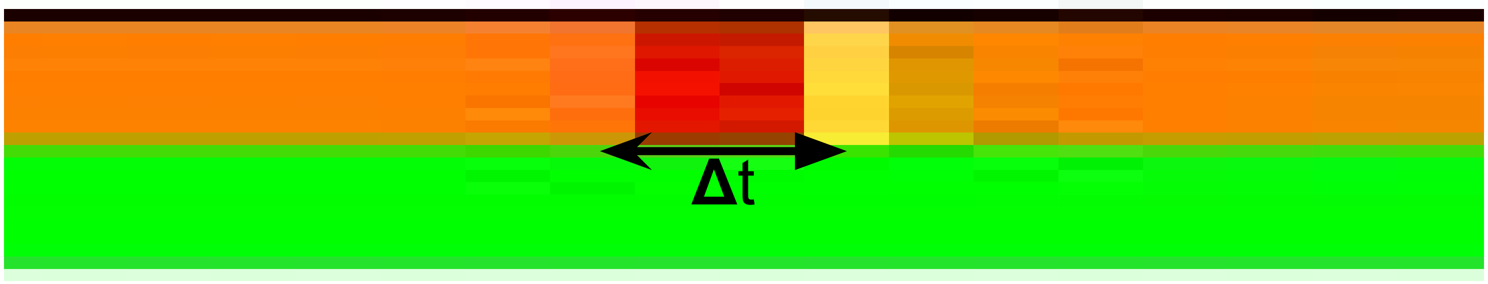
The cycle time of RT task being context-switched, entering CPU



PREEMPT_RT Cortex-A9 running cyclictest

Time delayed in Run Queue, waiting for scheduler to reschedule





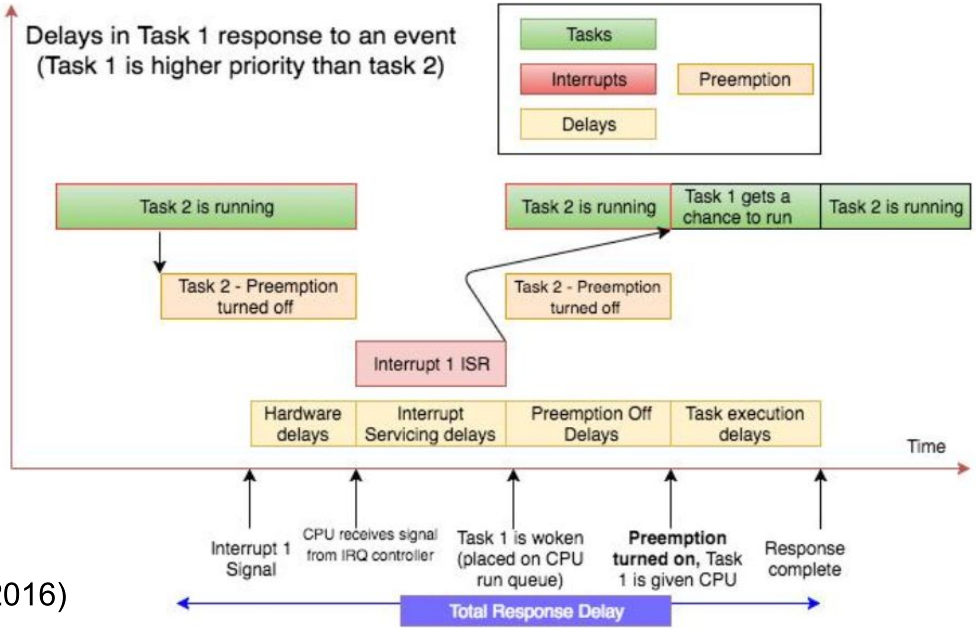
Reduce the Latency

Tips on PREEMPT_RT

- Preemption is disabled after acquiring raw_spinlock
  - Preemption off for long time is a problem (high prio task cannot run)
  - PREEMPT_RT makes critical sections preemptible
- When disable preemption (effect of locking CPU to other tasks), use need_resched() to check if higher priority task needs CPU to break out of preempt off section.
- Convert OSQ lock to atomic_t to reduce overhead

Linux mutex utilizes OSQ lock which will spin in some conditions with PREEMPT_RT.

optimistic spinning for sleeping



Source: Debugging Real-Time issues in Linux, Joel Fernandes (2016)

IRQ again

- IRQ threads are SCHED_FIFO tasks with priority 50.
  Priority can be changed, so that other RT tasks could have higher priority.
- Avoid unnecessary (raw_)spinlock_irq_save

```
static void atomisp_css2_hw_load(hrt_address addr, void *to, uint32_t n) {
    unsigned long flags;
    char *_to = (char *) to;
    spin_lock_irqsave(&mmio_lock, flags);
    raw_spin_lock(&pci_config_lock);
    for (unsigned i = 0; i < n; i++, _to++, _from++) *_to = _hrt_master_port_load_8(_from);
    raw_spin_unlock(&pci_config_lock);
    spin_unlock_irqrestore(&mmio_lock, flags)
}
```

spin_lock_irqsave does not disable interrupts in PREEMPT_RT_FULL.

```
// can be replaced with
disable_irq_nosync(irq);
spin_lock(&mmio_lock)
```

Deeper discussion in: Debugging Real-Time issues in Linux, Joel Fernandes (2016)

System Call Overhead

- System calls have almost universally been implemented as a synchronous mechanism, where a special processor instruction is used to yield userspace execution to the kernel.
- FlexSC implements exceptionless system calls in Linux kernel, and an accompanying user-level thread package (binary compatible with PThread), that translates legacy synchronous system calls into exception-less ones transparently to applications.
- FlexSC improves performance of Apache by up to 116%, MySQL by up to 40%, and BIND by up to 105% while requiring no modifications to the applications.

Eliminate latency to enter system call

- Kernel Mode Linux (KML): Execute user processes in kernel mode
- Benefit of executing user programs in kernel mode is that the user programs can access a kernel address space directly.

  user programs can invoke system calls very fast because it is unnecessary to switch between a kernel mode and a user mode by using costly software interruptions or context switches.
- Unlike kernel modules, user programs are executed as ordinary processes (except for their privilege level), so scheduling and paging are performed as usual.

  Although it seems dangerous to let user programs access a kernel directly, safety of the kernel can be ensured, for example, by static type checking, software fault isolation, and so forth.

Case Study:
ARM Cortex-A9 MP

Experimental Platforms

- Altera Cyclone V SoC Development Kit
  - CPU : ARM Cortex-A9 Dual Core
  - Memory : 1 GB DDR3

- NXP i.MX6Q Sabre SDB
  - CPU : ARM Cortex-A9 Quad Core
  - Memory : 1 GB DDR3

Experiment Configurations

- Buildroot based System
- Linux Kernel 4.4 with PREEMPT_RT,
- Optional additional patches:
  - Wasted Cores Patches
  - Kernel Mode Linux

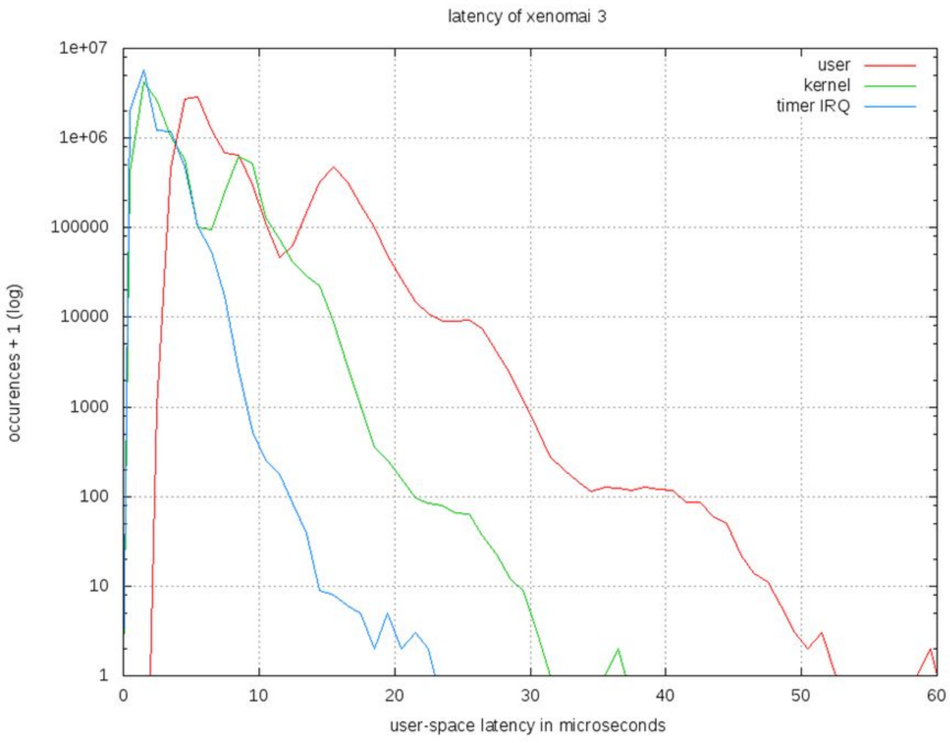
Benchmark Suite

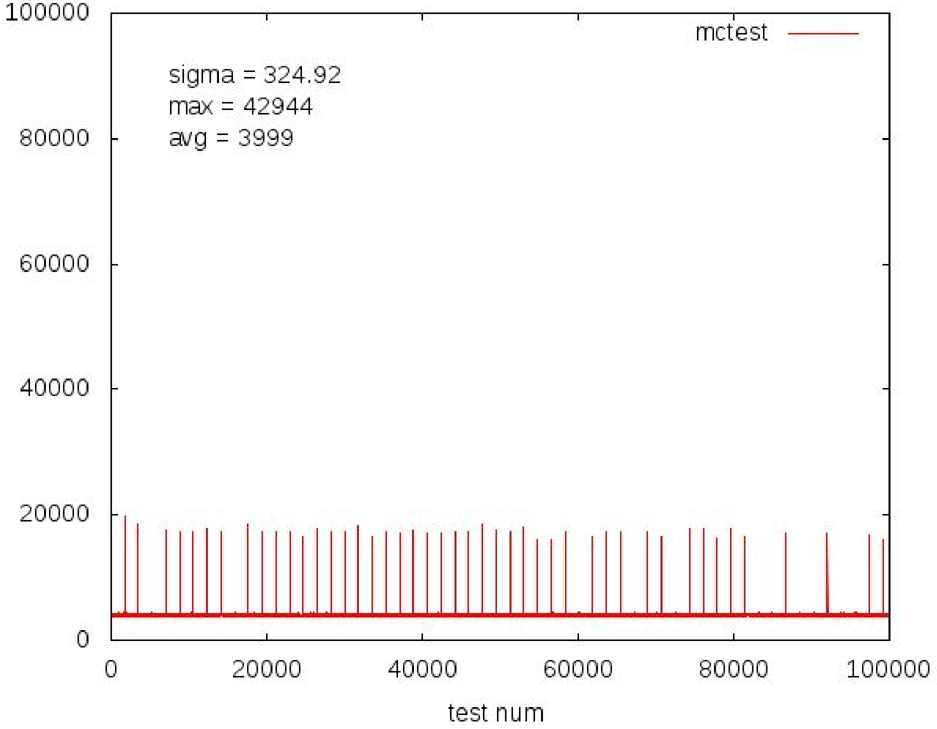
- RT test bench:
  - Cyclictest form rt-tests
    - cyclictest -mnq -p 90 -h 1000 -i 1000 -l 1000000
    - Run in KML if KML is enabled
  - Mctest
    - User-Space Program
      - Run in KML if KML is enabled
    - Kernel-Space Kernel Module

Experiment Test Benches

- Mctest
  - Measuring the determinism of code execution time
  - Developed to simulate Robot Motion Contol Algorithm
  - Could execute as
    - User-space program
    - Kernel module in Kernel Space
  - Outputs
    - The execution time of each run

Experiment Test Benches





Experiment Setup

- Loads and arguments used
  - Hackbench
    - hackbench -s 512 -l 1024 -P
  - Stress
    - stress --cpu 3 --timeout=10
    - stress --cpu 8 --timeout=10
    - stress --cpu 4 --io 2 --vm 2 --vm-bytes 128M --timeout=10
  - Mctest, User-space
  - Mctest, Kernel-space
  - Netperf

Experiments and Measurements

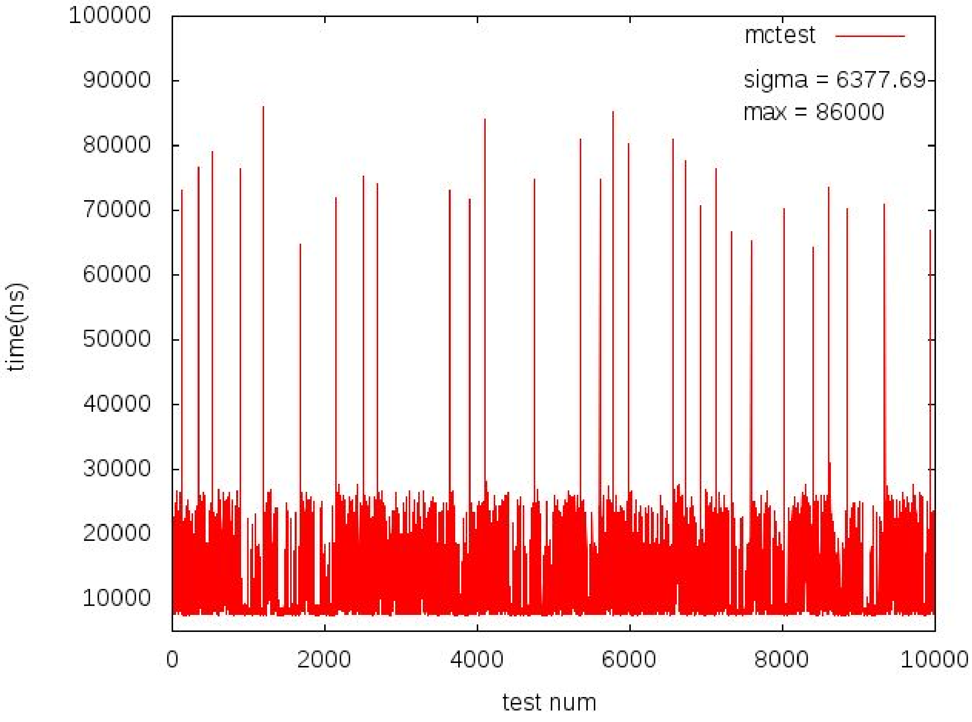
- Kernel Mode Linux’s impact on real-time performance
- SMP schedulability
  - Unbalanced Workload
  - RT Wake-up of Overloaded Core
  - KML’s Impact to Scheduler
- Short Inter-arrival Time
- Scheduler Duration

Kernel Mode Linux's impact on real-time performance

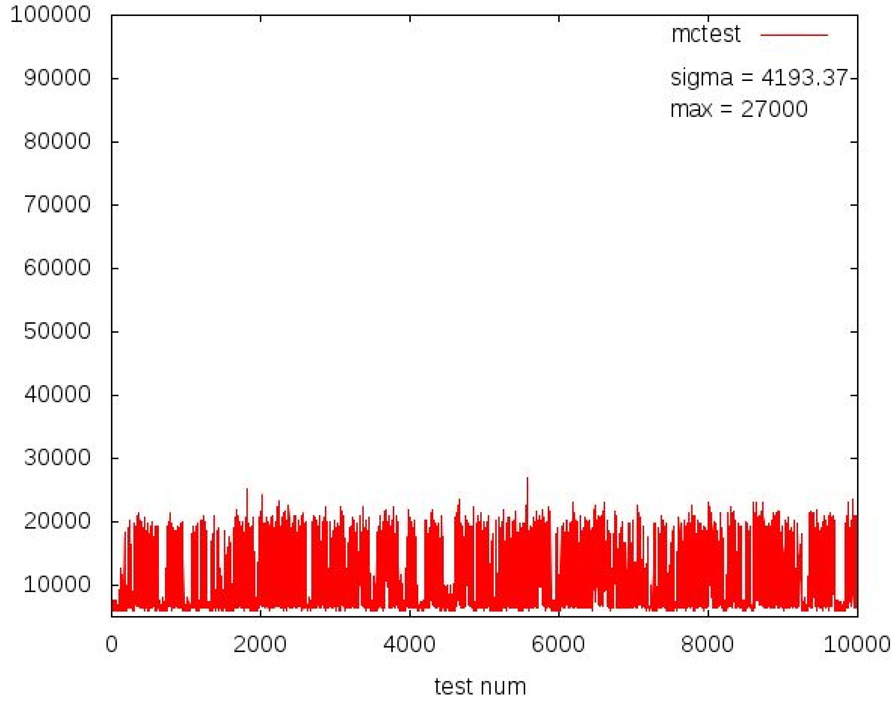
- The following test is running with
  - i.MX6 sabre SDB
  - CPU 1 isolated and set as tickless
  - L2 Cache Locked Down to CPU 1
  - Load is in combination of:
    - Hackbench
    - Netperf
  - Test Bench: mctest

Kernel Mode Linux's impact on real-time performance

User-Space Mctest



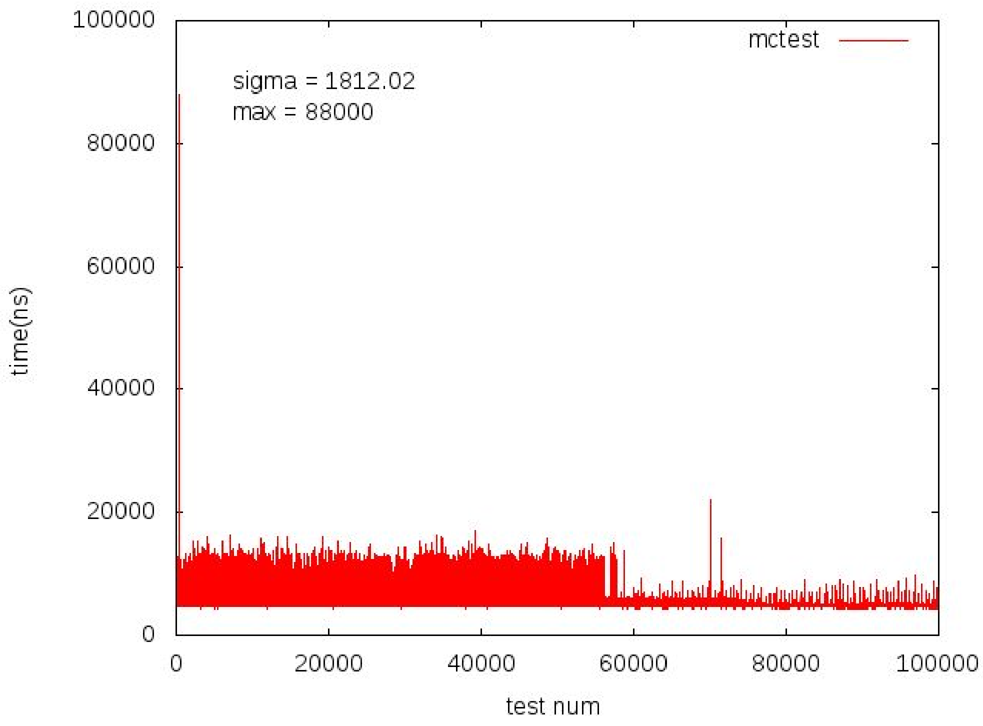
Kernel-Space Mctest



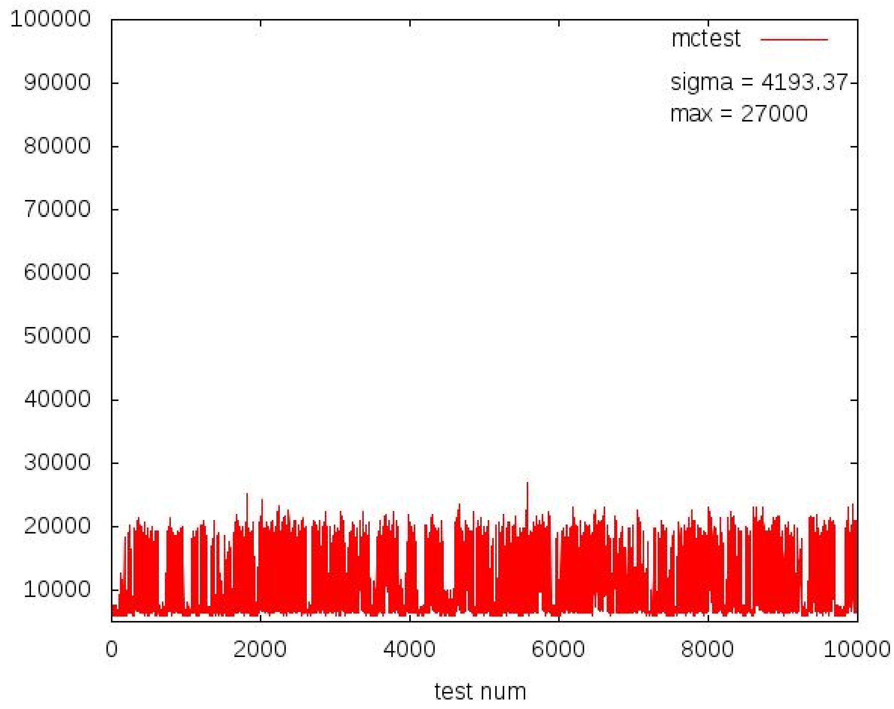
- Without Kernel Mode Linux
  - The imapct from system calls are high
  - Result has a lot of spikes

Kernel Mode Linux's impact on real-time performance

User-Space Mctest in KML



Kernel-Space Mctest

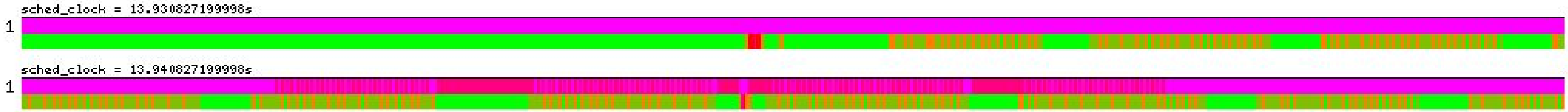


- Kernel Mode Linux
  - Siginificant reduce impact from system calls
  - Result is comparable against Kernel-Space Mctest

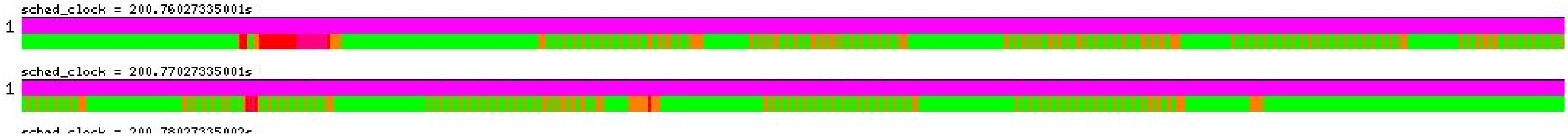
SMP schedulability - Unbalanced workload

Stress (4 CPU, 2 IO, 2 VM=128M) on Cyclone V SoC

Cyclone V SoC, PREEMPT_RT (1px = 10us)



Cyclone V SoC, PREEMPT_RT + Wasted Cores Patch (1px = 10us)

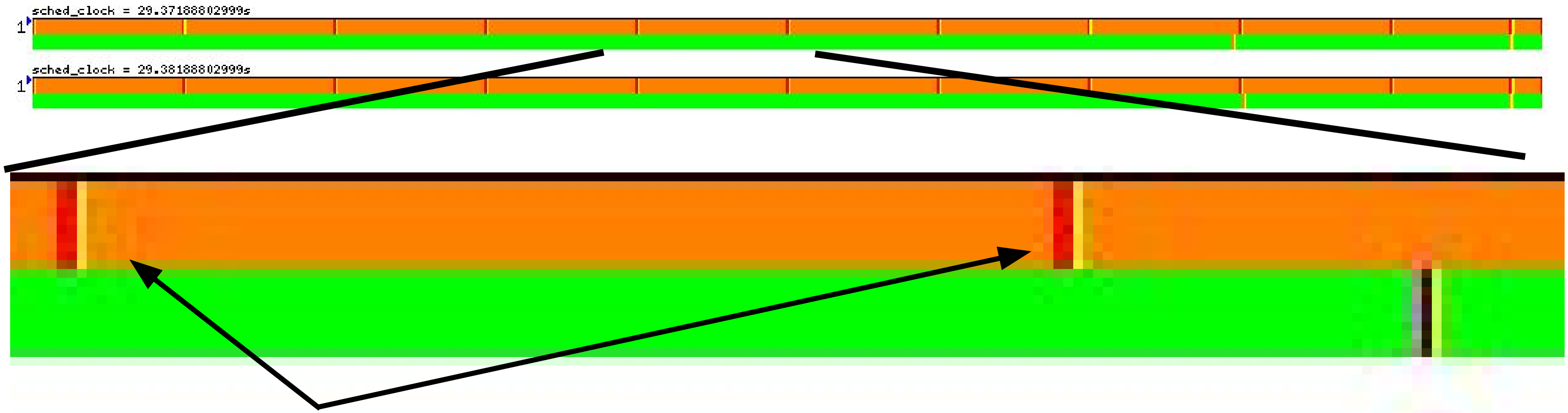


- By default, Linux Scheduler balances load every 10ms
- Thus, short burst, which < 10ms , will not be balanced
- Wasted Cores won’t help this kind of case

SMP schedulability - Wake-up on overloaded

Stress (3 CPU) + Cyclictest (1ms) on Cyclone V SoC

Cyclone V SoC, PREEMPT_RT (1px = 1us)



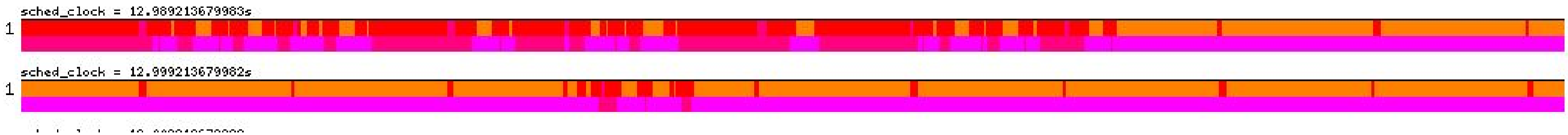
- Short burst RT task will be scheduled on overoaded cores
- This short burst of unbalance won't harm long term throughput
- But could cause impact to the RT performance

SMP schedulability - KML's impact

Stress (8CPU) on Cyclone V SoC

PREEMPT_RT (1px = 1us)

PREEMPT_RT + Wasted Cores Patch + KML (1px = 1us)

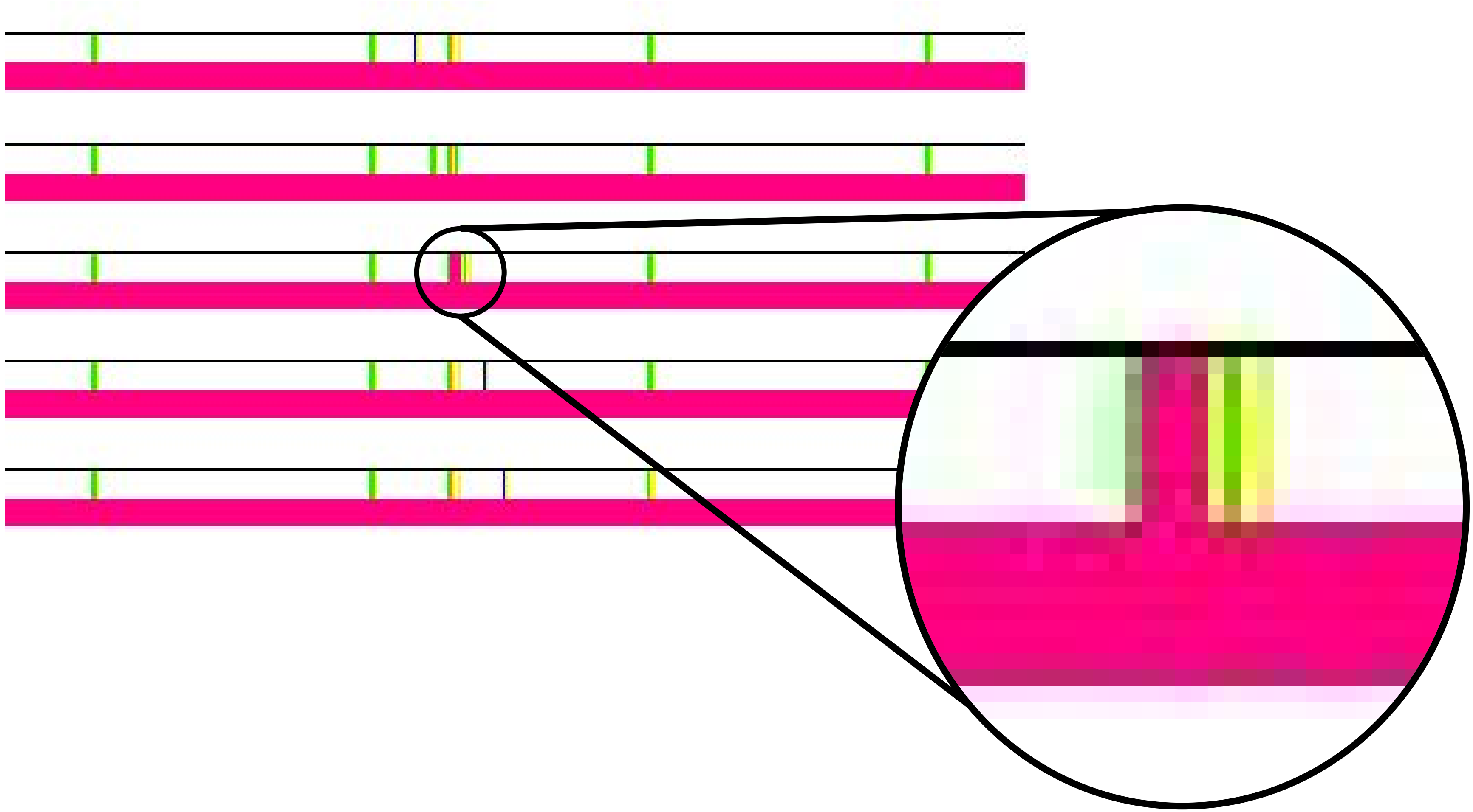


- KML reduces system call overhead
- Thus, no impact on the scheduler behavior and latency

Short Inter-arrival Time - Timer IRQ against Cyclictest's main Task

Mctest Kernel + Cyclictest (1ms) on Cyclone V SoC

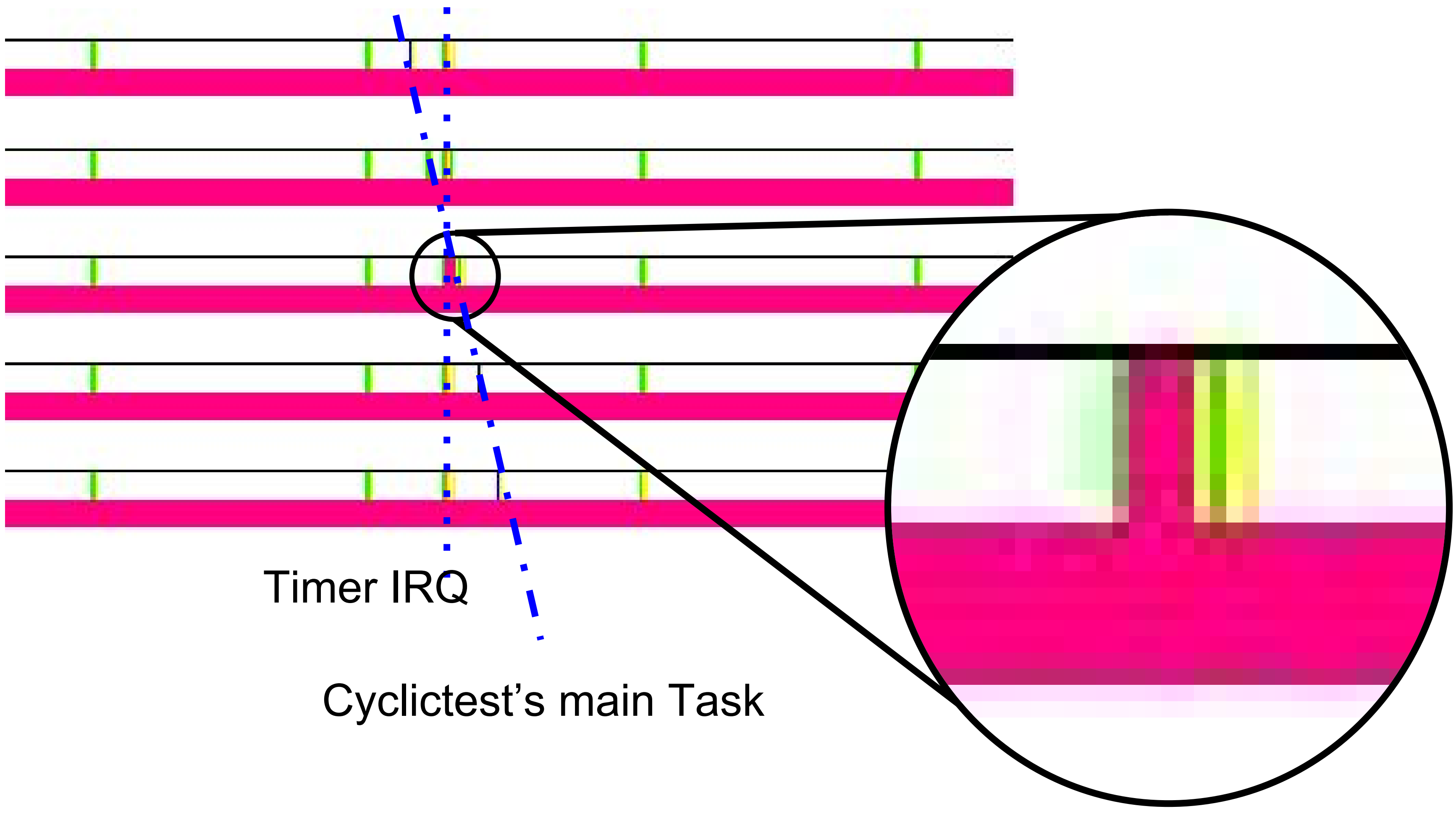
PREEMPT_RT (1px = 10us)



Short Inter-arrival Time - Timer IRQ against Cyclictest's main Task

Mctest Kernel + Cyclictest (1ms) on Cyclone V SoC

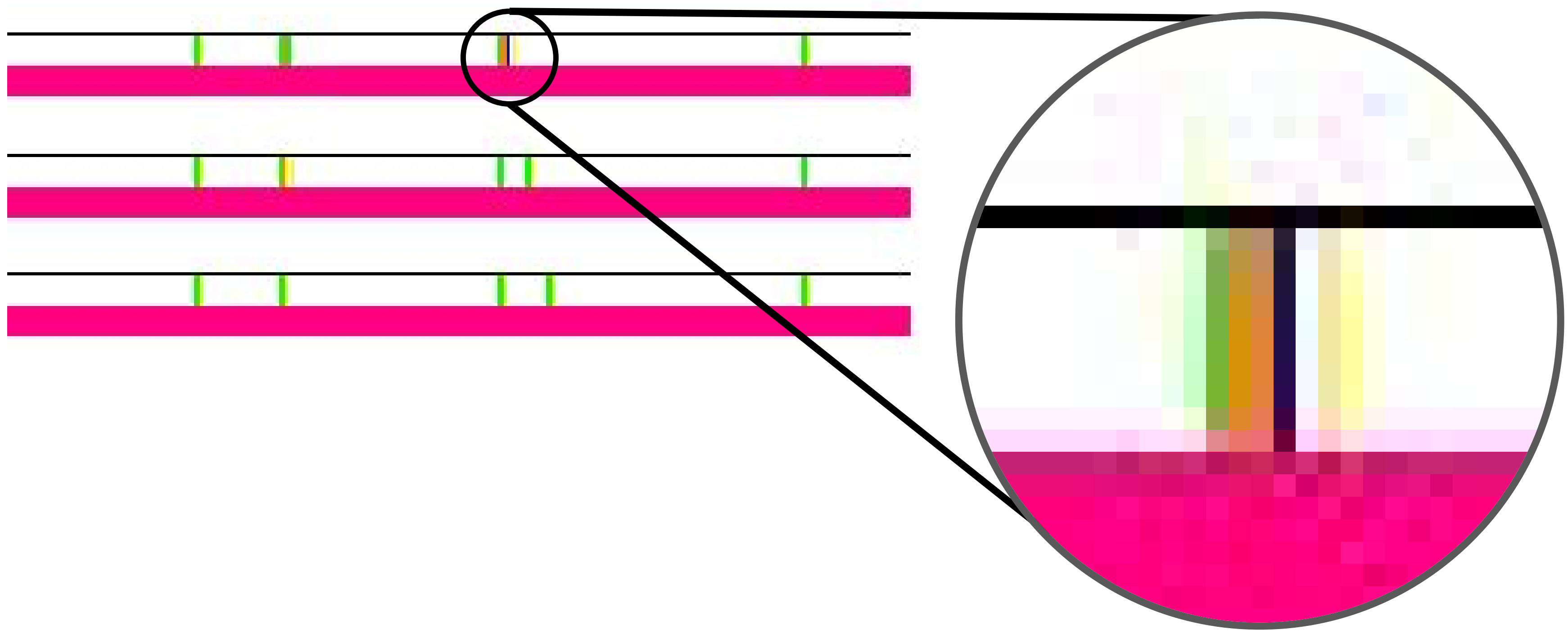
PREEMPT_RT (1px = 10us)



Short Inter-arrival Time - RT Task against Cyclictest's main Task

Mctest Kernel + Cyclictest (1ms) on Cyclone V SoC

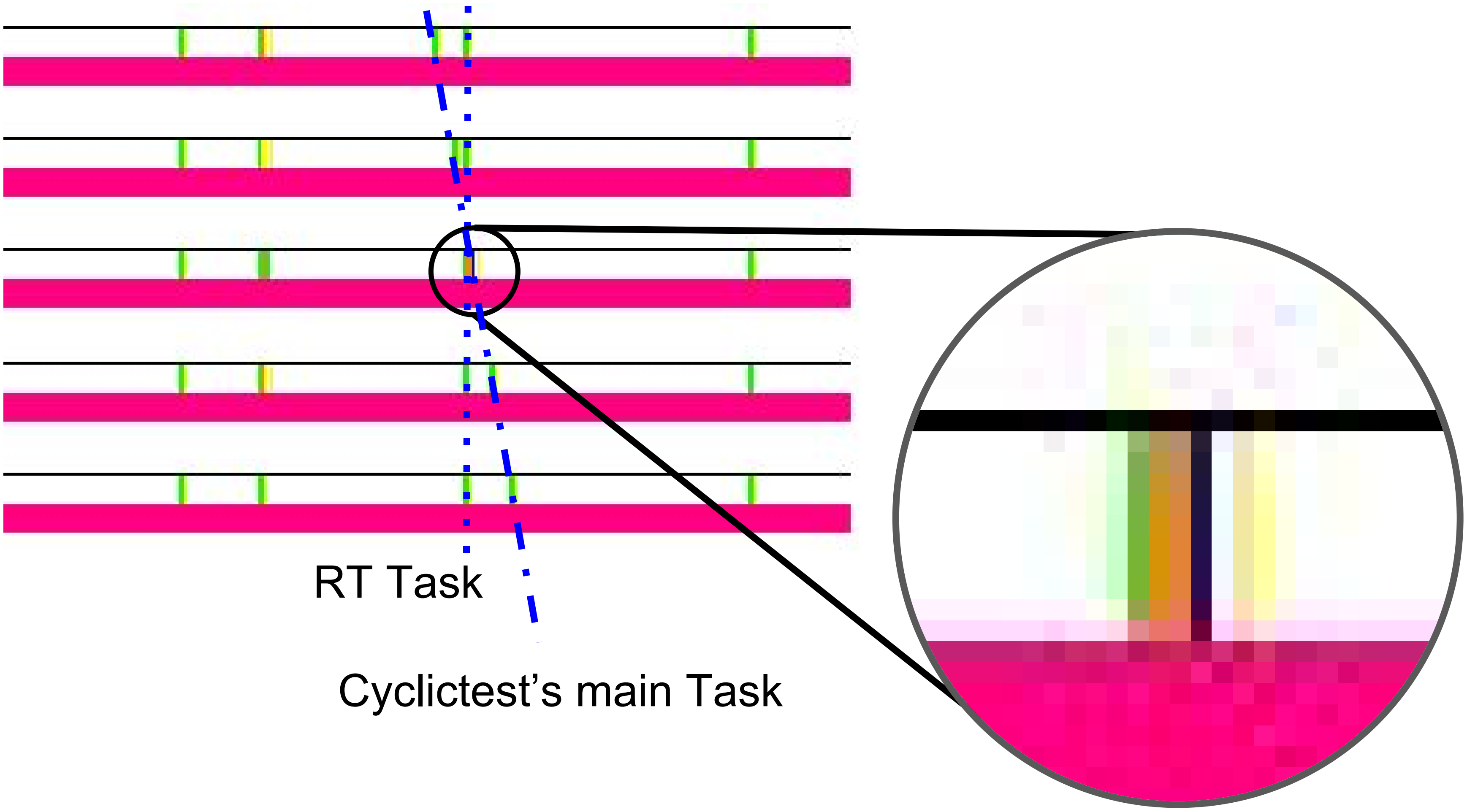
PREEMPT_RT (1px = 10us)



Short Inter-arrival Time - RT Task against Cyclictest's main Task

Mctest Kernel + Cyclictest (1ms) on Cyclone V SoC

PREEMPT_RT (1px = 10us)



Short Inter-arrival Time

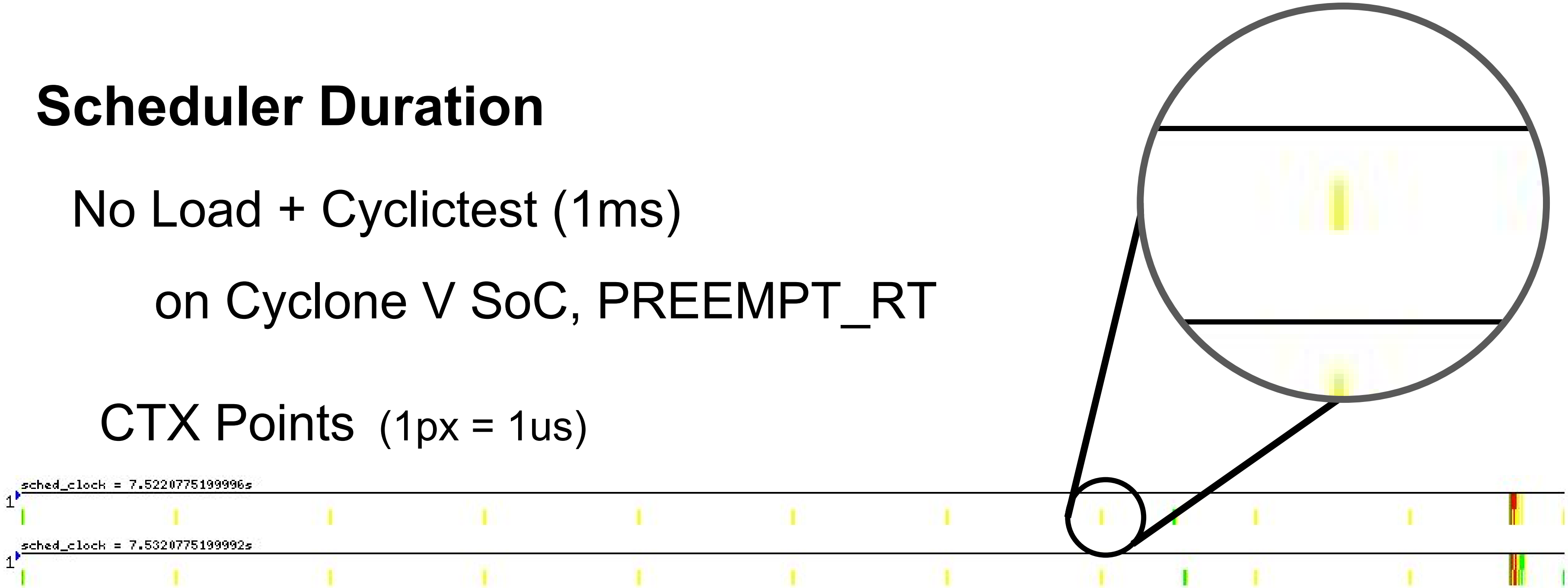
- Can cause IRQ Bottom halves delay
- Can cause cost of scheduling raise
- Would be harmful to the real-time performance

Scheduler Duration

No Load + Cyclictest (1ms)

on Cyclone V SoC, PREEMPT_RT

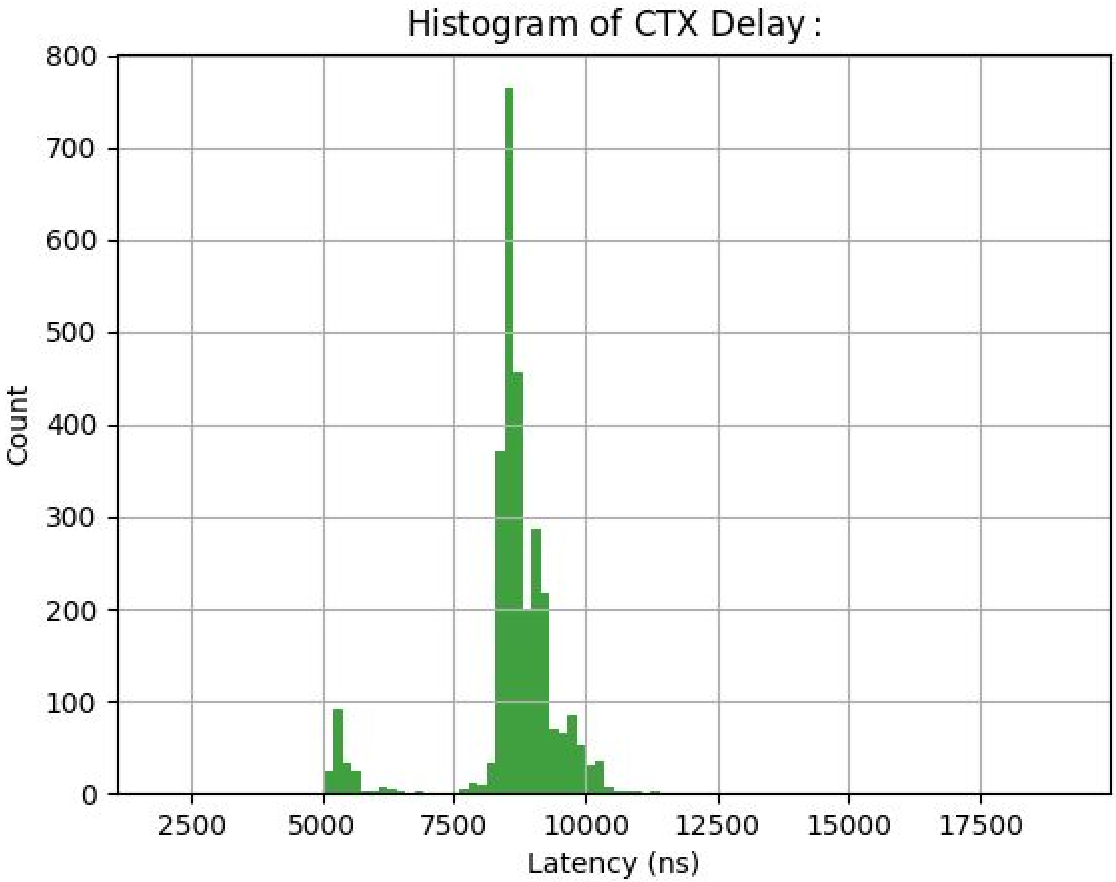
CTX Points (1px = 1us)



Delay from Entering RQ to CTX of each run Vertical:(1px = 0.1us)

Wake-up Latency of Scheduler

No Load + Cyclictest (1ms) on Cyclone V SoC, PREEMPT_RT



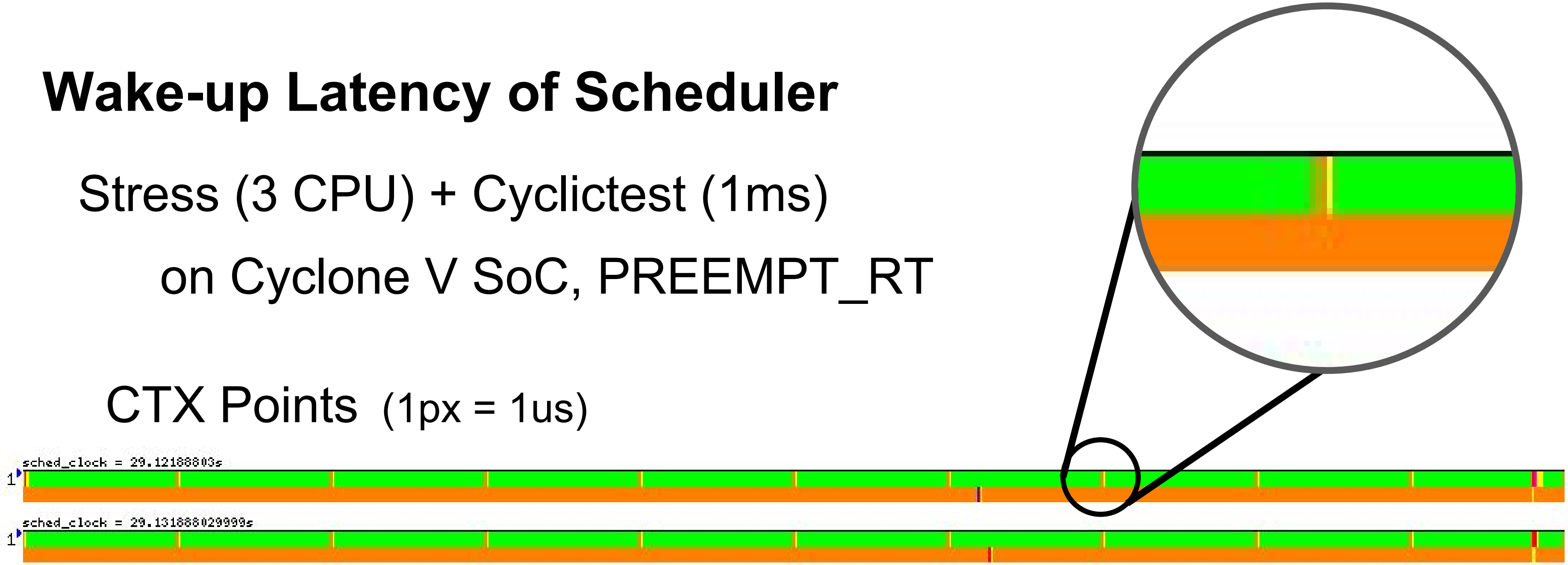
Max: 20us

Wake-up Latency of Scheduler

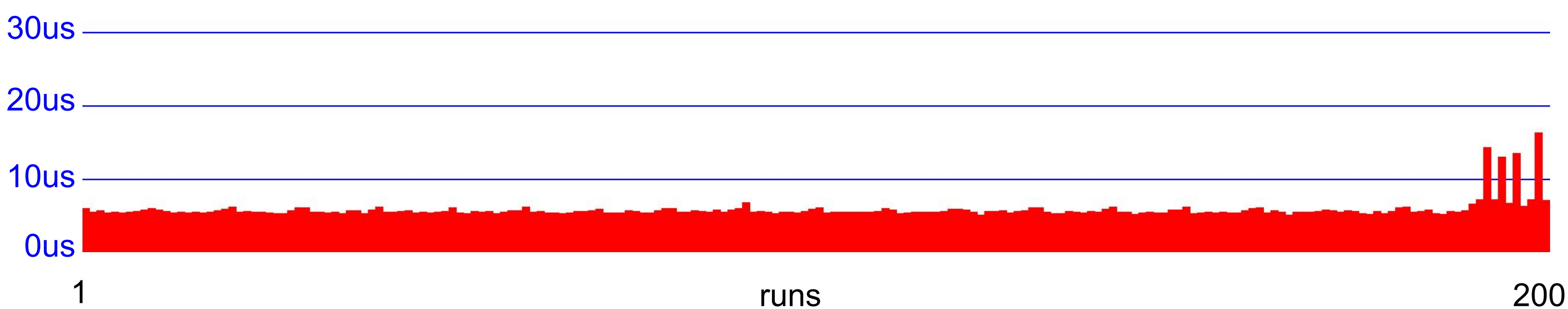
Stress (3 CPU) + Cyclictest (1ms)

on Cyclone V SoC, PREEMPT_RT

CTX Points (1px = 1us)



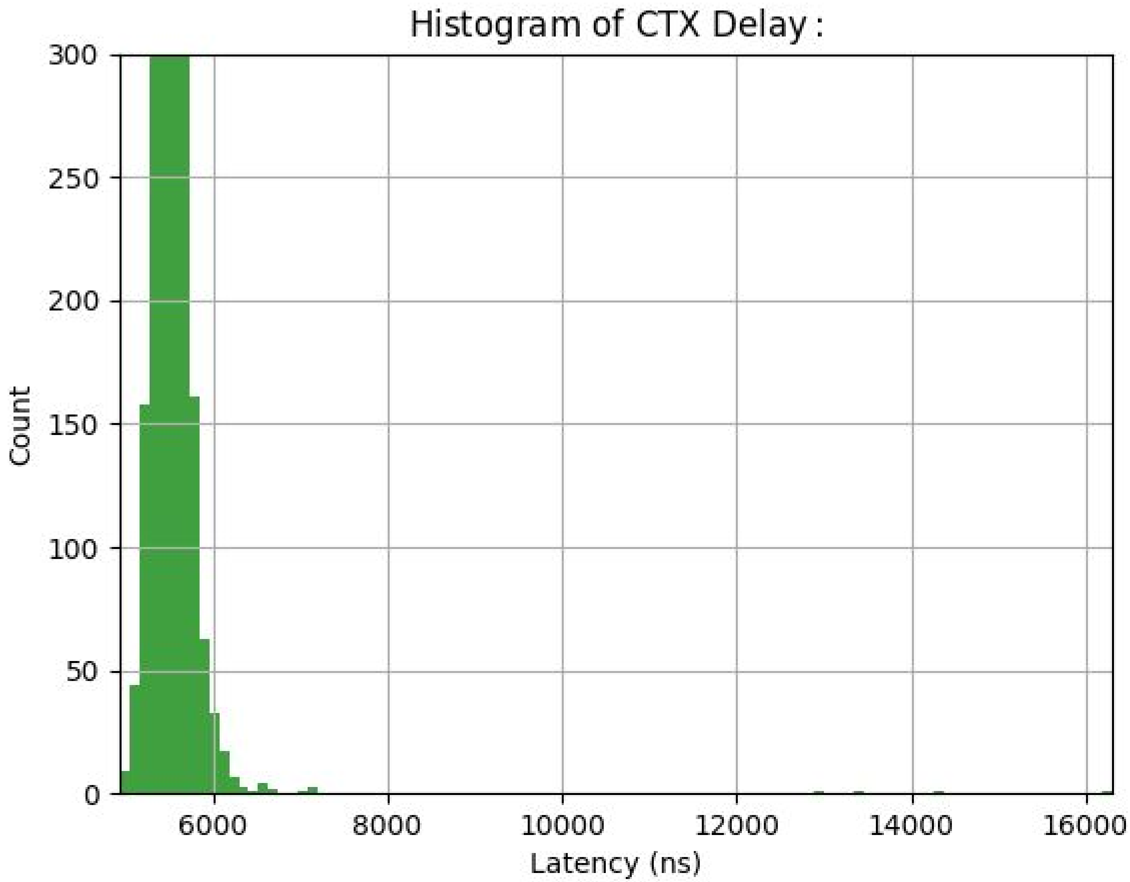
Delay from Entering RQ to CTX of each run Vertical:(1px = 0.1us)



Wake-up Latency of Scheduler

Stress (3 CPU) + Cyclictest (1ms)

on Cyclone V SoC, PREEMPT_RT



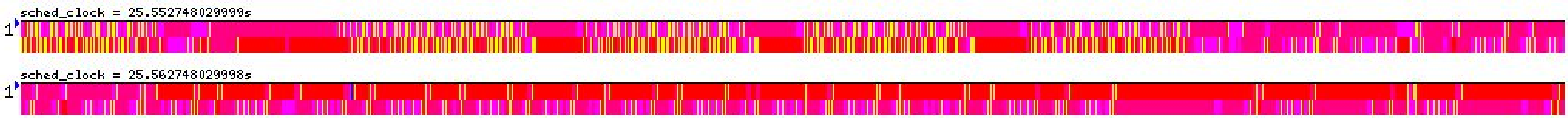
Max: 16us

Wake-up Latency of Scheduler

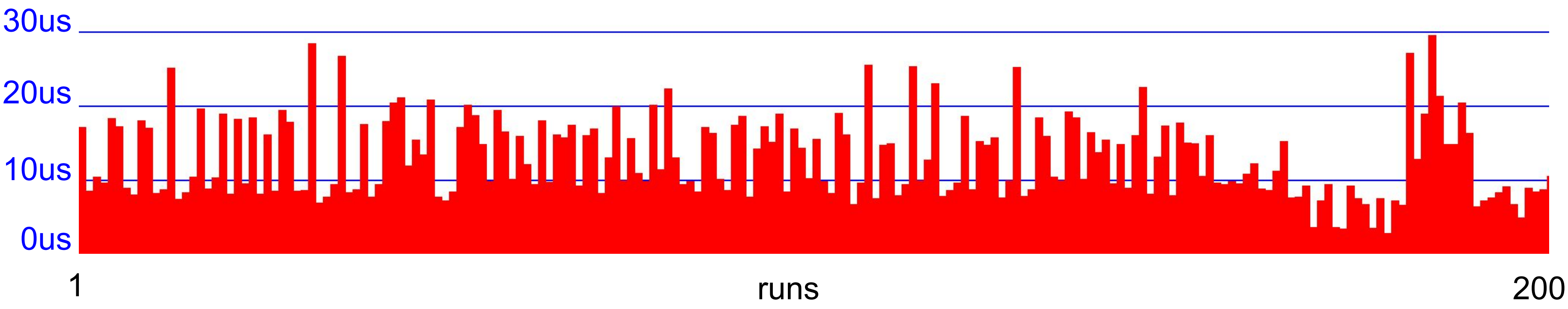
Stress (8 CPU) + Cyclictest (1ms)

on Cyclone V SoC, PREEMPT_RT

CTX Points (1px = 1us)



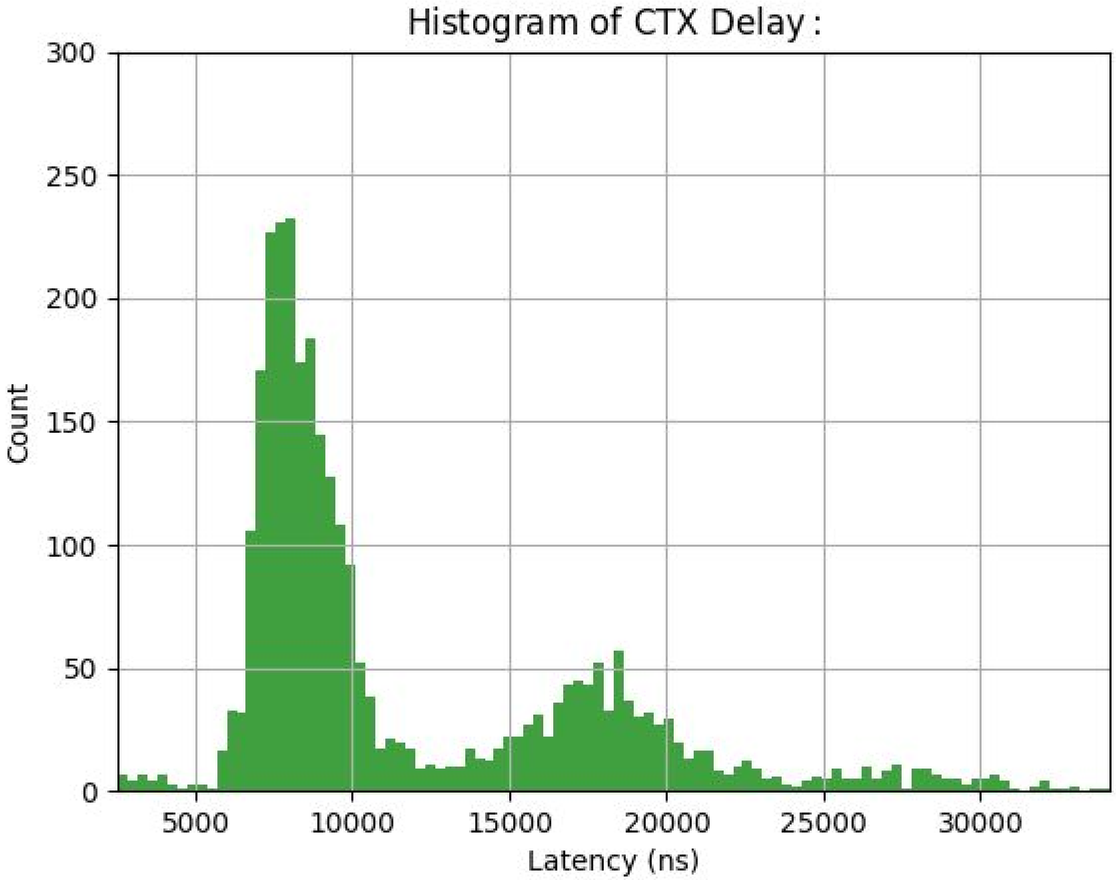
Delay from Entering RQ to CTX of each run Vertical:(1px = 0.1us)



Wake-up Latency of Scheduler

Stress (8 CPU) + Cyclictest (1ms)

on Cyclone V SoC, PREEMPT_RT

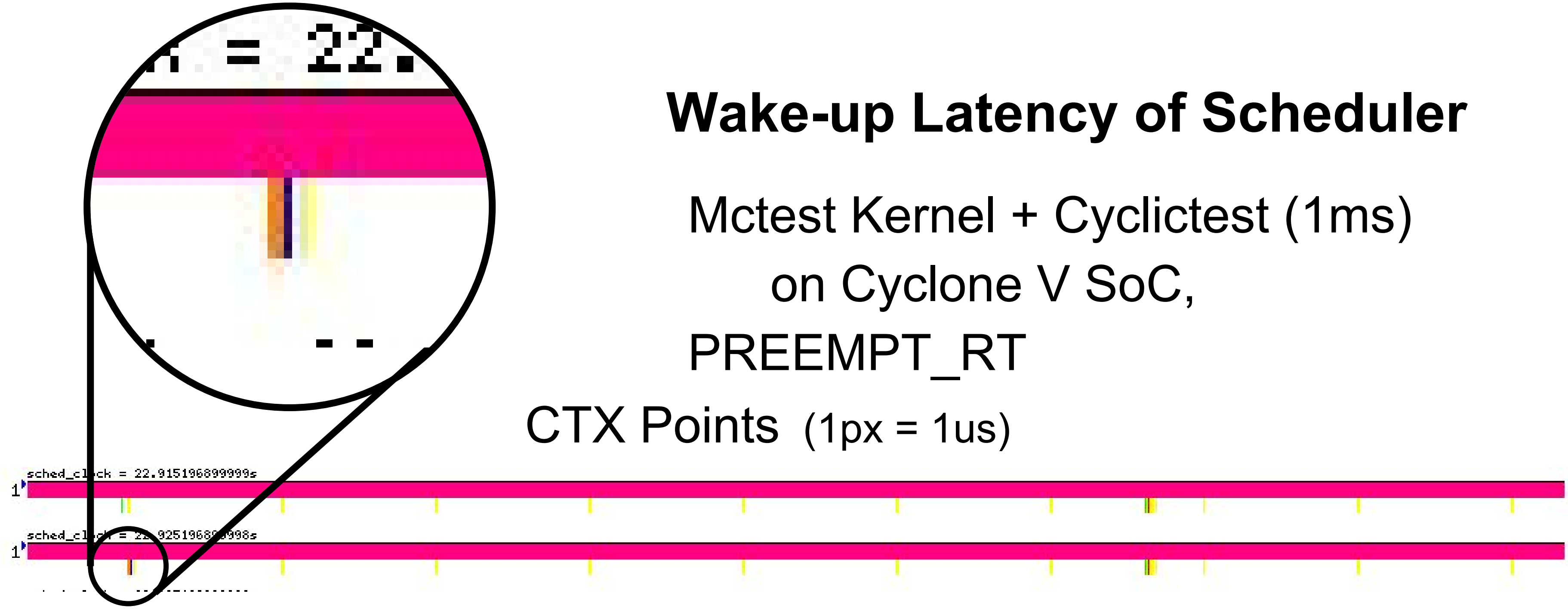


Max: 34us

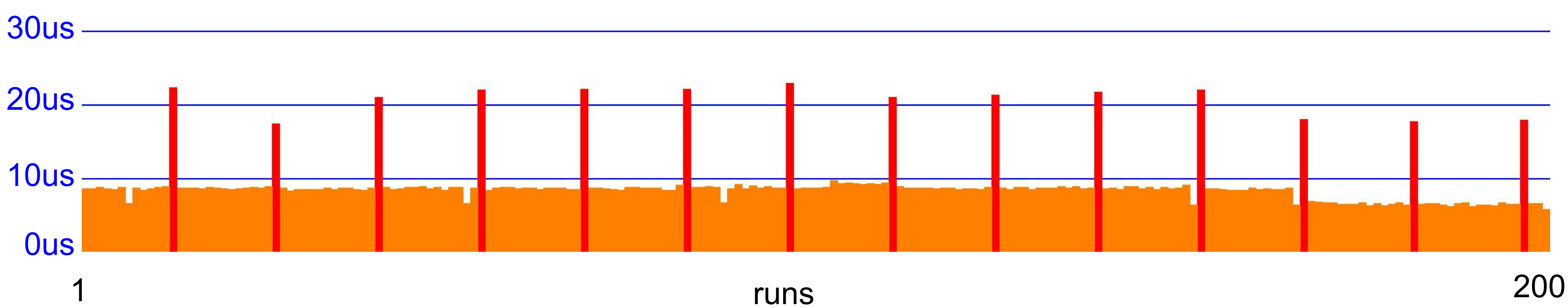
Wake-up Latency of Scheduler

Mctest Kernel + Cyclictest (1ms)
on Cyclone V SoC,
PREEMPT_RT

CTX Points (1px = 1us)

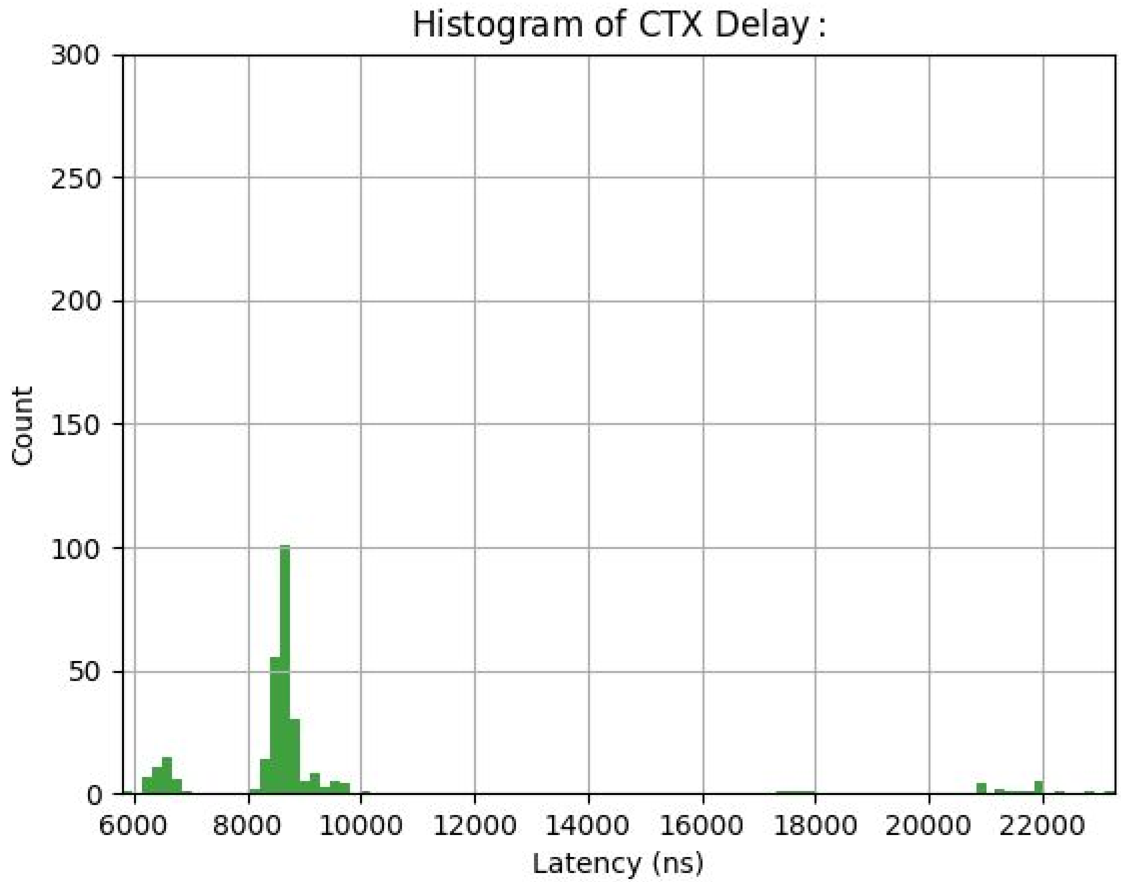


Delay from Entering RQ to CTX of each run Vertical:(1px = 0.1us)



Wake-up Latency of Scheduler

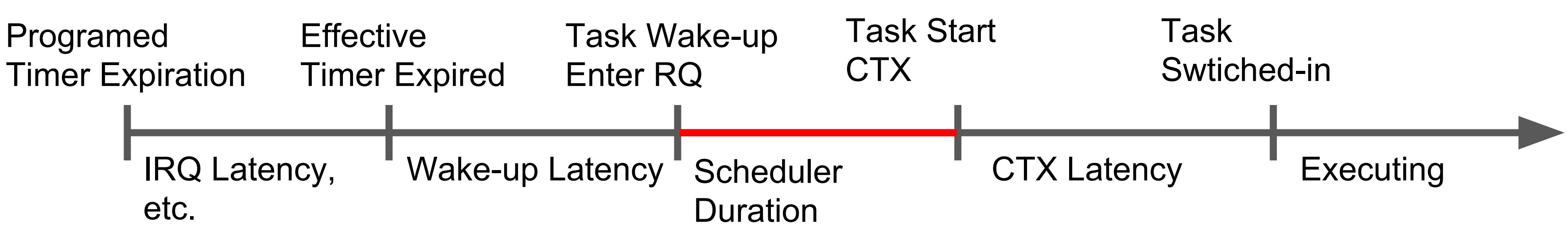
Mctest Kernel + Cyclictest (1ms)
on Cyclone V SoC, PREEMPT_RT



Max: 23us

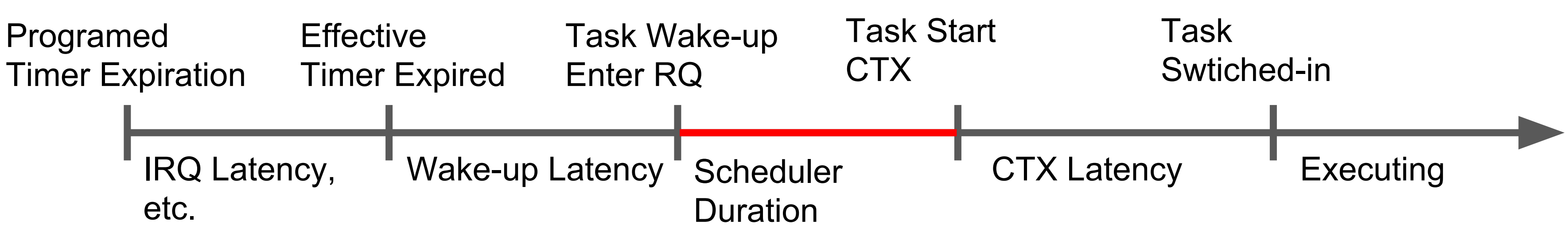
Observation of Scheduler Duration

- When scheduler enqueued a high priority task into run queue, it would require a period of scheduler duration before switching it in for execution.
- Scheduler duration between entering RQ and CTX would be at most 35us, depends on load.



Observation of Scheduler Duration

- Scheduler grantees O(1) on searching
- After identifying the next task for executing, scheduler would still spend extra time, which would vary with the load in scheduling run queue.
- The shorter the inter-arrival time, the larger the scheduler duration distribution spreads.



Our contribution for Real-time system measurements and enhancements

- Kernel configs, Buildroot configs, and misc: https://github.com/sonicyang/rt-experiments
- Kernel Mode Linux (KML): https://github.com/sonicyang/KML
- Mctest: https://github.com/sonicyang/mctest
- WastedCores Patches: https://github.com/sonicyang/wastedcores

Conclusion

- We have evaluated the real-time behavior of Linux by profiling kernel scheduler and measuring the latency of various kernel variants.
- An intensive interrupt load can cause long OS latencies due to the design of the interrupt processing mechanism. We proposed new tools to visualize task scheduling in fine-grained scale(microsecond level). This enabled us not only focusing on interrupt latency, but also scheduler durations, lock, and etc.
- It would thus be highly desirable to combine existing techniques, e.g KML, isolated CPU, tickless kernel, to improve task responsiveness under various target application characteristics, on top of PREEMPT_RT.

Reference

- Understanding a Real-Time System, Steven Rostedt
- Evaluation of Real-time Property in Embedded Linux, Hiraku Toyooka, Hitachi
- Real-time Throughput, Gregory Haskins & Steve Rostedt
- An Essential Relationship between Real-time and Resource Partitioning, Yoshitake Kobayashi, TOSHIBA
- A Decaded of Wasted Cores, Jean-Pierre Lozi, et al. (EuroSys 2016)
- FlexSC: Flexible System Call Scheduling with Exception-Less System Calls, Livio Soares & Michael Stumm (OSDI 2010)